Department di Informatica
Università degli Studi di Salerno

Dependability analysis on Commons-Fileupload

May 25th, 20223

Project Report

Aiman Faiz (a.faiz@studenti.unisa.it)
Supervised by:
Prof: Dario Dinucci

Table of Contents

Chapter 1: Introduction

This report is about the different dependability analysis which I ran on Apache software foundation project named "Commons-file upload".

1.1. Commons-Fileupload

Apache Commons FileUpload is a popular Java library that provides a simple and robust solution for handling file uploads in web applications. The Commons FileUpload package simplifies the process of incorporating reliable and efficient file upload functionality into servlets and web applications.

Apache Commons FileUpload simplifies the process of handling file uploads in Java web applications. It provides a comprehensive set of features, flexibility, and wide compatibility, making it suitable for a variety of web application scenarios.

1.2. Report Structure

This report covers all the analysis and procedures which were used to measure the reliability, dependability, and security of this project.

The first chapter starts with a little introduction about the project itself and explains the scope of this report. This chapter also includes information about the external software and tools which were used during the building of the system, measuring the software quality, and generating new test cases.

The second chapter is the heart of this report which explains all the analysis and procedures in detail along with their results and changes which were made by the analysis. It includes building of the system, then orchestration of the project via Docker and Docker Hub, Quality analysis with sonar, code coverage via JaCoCo, Mutation testing, Analysis of energy greediness by EcoCode, Performance testing with JMH, generation of Regression test cases and measuring security aspects of the project.

The third chapter covers some code blocks and additional plugins which were used during the project. Fourth and Fifth chapter shows conclusions and references respectively.

1.3. *External tools and software*

1.3.1. Tools and software

For building the analysis, I needed certain tools and building systems. For cloning and storing my repository git/GitHub was used. I used IntelliJ as my IDE (Integrated development environment). As part of this project included the containerization of the project, for that I used Docker and for pushing my project image on docker repository I used Docker hub.

1.3.2. Building the system

For building the system I used Maven version 3.9.1. I also used JDK 17 in some analysis and JDK 8 in some others. I used JDK because during the building of a project, Maven, without toolchains, used JDK to perform various steps, like compiling the Java sources, generating the Javadoc, running unit tests, or sign JARs. JDK provides the necessary tools and libraries for compiling, building, and running Java applications.

By utilizing JDK, Maven ensured that the necessary Java development tools and runtime environment are available for building and managing projects effectively.

Chapter 2: Software Dependability Analysis

2.1. Building the Project

This project was started by forking the project in my GitHub (https://github.com/Aimanfaiz1999/commons-fileupload) and then locally building the project. Project was locally cloned on my machine by git clone command after forking on GitHub using git bash terminal. For editing a running new analysis, I used IntelliJ and in IntelliJ the project was built with Maven 3.9.2. I set up relevant workflows and connected my IntelliJ with my GitHub so all the commits and changes could work.

2.2. SonarCloud Analysis

I used SonarCloud analysis for checking code quality and security. It utilizes static code analysis and detects bugs, vulnerabilities, and code smells in my projects.

In my initial check with sonar cloud, I obtained following results:

As per my severity level I had 8 critical issues, 28 minor issues and 26 major issues.

I refactored 3 critical issues, 19 major issues, 23 minor issues and solved 3 bugs respectively.

2.2.1. Findings and Refactoring

a. Critical issues:

i. This issue was found in src/main/java…/commons/fileupload2/AbstractFileUpload.java. I reduced methods Cognitive Complexity from 18 to 15. Cognitive Complexity measures how hard is to control flow of a method to understand. High Cognitive Complexity of methods make them difficult to maintain.

ii. This refactoring took place in src/main/java/… fileupload2/MultipartStream.java. I defined a constant instead of duplicating the literal "Stream ended unexpectedly" 3 times to decrease duplication. The new constant is named as MY_CONSTANT

b. Major issues:

i. This refactoring took place in src/test/java/org/apache/commons/fileupload2/ MultipartStreamTest.java at line 41. I refactored the code of the lambda to have only one invocation possibly throwing a runtime exception.

Before: assertThrows(IllegalArgumentException.class,
() -> new MultipartStream(
input,
boundary,
iBufSize,
new MultipartStream.ProgressNotifier(null, contents.length)));
After: assertThrows(IllegalArgumentException.class, () -> {

```
    MultipartStream.ProgressNotifier progressNotifier = new
MultipartStream.ProgressNotifier(null, contents.length);
    new MultipartStream(input, boundary, iBufSize, progressNotifier);
});
```

ii. This refactoring took place in src/test/java/org/apache/commons/fileupload2/ FileItemHeadersTest.java at lines 56,59,60,61,69,73,78,80,82 and 84. I swapped the 2 arguments, to make them in correct order: expected value, actual value.

i.e. assertEquals(expected,name)

iii. This refactoring took place in src/main/java/org/apache/commons/fileupload2/ AbstractFileUpload.java. On line 158 "if" block was returning a null array instead of empty.

iv. This refactoring took place at src/main/java/../impl/FileItemIteratorImpl.java. I removed the unused method parameter named "requestContext" for saving the resources at line 251.

v. This issue was found in src/main/java…/commons/fileupload2/util/LimitedInputStream .java. I changed the scope of constructor from public to protected

c. Minor issues:

i. This is the issue of declaration of thrown exception. I removed the declaration of thrown exception in following files because as their exceptions were subclass of 'java.io.IOException

- src/main/java…/commons/fileupload2/MultipartStream.java
- src/main/java…/commons/fileupload2/Abstractfileupload.java
- src/main/java…/commons/fileupload2/FileItemIterator.java
- src/main/java…/commons/fileupload2/FileItem.java
- src/main/java…/commons/fileupload2/disk/DiskFileItem.java
- src/main/java…/commons/fileupload2/imp/Fileitemiterator.java
- src/main/java… /commons/fileupload2/servlet/ServletFileUpload.java
- src/main/java… /commons/fileupload2/portlet/portletFileUpload.java
- src/main/java… /commons/fileupload2/util/mimi/limitedinputstream

ii. This issue was found in src/main/java…/commons/fileupload2/disk/DiskFileItem. I replaced the "switch" statement by "if" statements in order to increase its readability.

iii. This issue was found in src/main/java…/commons/fileupload2/disk/DiskFileItem. Removed transient modier in fields.

iv. This refactoring took place in src/main/java/….java Refactored the code at line 43 to make use of more specialised Functional Interface 'UnaryOperator<String>'.

v. I changed the scopes of test cases from public to none in following test classes:

- src\test\java…\fileupload2\util\mime\MimeUtilityTestCase
- src\test\java…\fileupload2\util\mime\QuoteddPrintableDecoderTestCase
- src\test\java…\fileupload2\serverlet\ServerletFileUploadTest
- src\test\java…\fileupload2\portlet\portletFileUploadTest
- src\test\java….\fileupload2\StreamingTest.java
- src\test\java….\fileupload2\SizesTest.java
- src\test\java….\fileupload2\progresslistnertest.java
- src\test\java….\fileupload2\parameterparsertest.java
- src\test\java….\fileupload2\multipartstream.java
- src\test\java….\fileupload2\fileuploadtest

- src\test\java....\fileupload2\fileitemheaderstest
- src\test\java....\fileupload2\diskfileitemserliezetest
- src\test\java....\fileupload2\jaksvltfileuploadtest
- src\test\java....\fileupload2\RFC2231UtilityTestCase.java

d. Skipped refactoring:

i. I did not change the class scope of many files because changing their public scope will make them unaccessible in other classes.

ii. There was a refactoring of for loop at line 101 in src/main/java../util/mime /RFC2231Utility.java It was about not to assign this loop counter from within the loop body but instead in the for loop statement. When i applied changes two of my tests were broken therefore I rolled back the changes.

iii. In src/main/java/…/pub/FileUploadContentTypeException.java at line 36 there was a string declared as privete named as contentType, I had to make it final and when I refactored it was unable to be inilize further in the method. Therefore I skipped it.

iv. I do not have enough knowledege of refactoring of some methods in a way to reduce their cogonitive complexities.

In my final check with sonar cloud, I obtained following results:

2.3 Docker orchestration and Image building

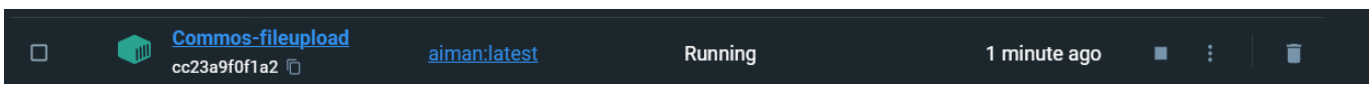
I used Docker for my project containerization. Docker is an open-source platform that enabled me to package, distribute, and run my software in isolated containers.

2.3.1. Image building

For image building, I installed Docker and set up my Docker hub account for pushing my image to the hub.

In addition, I added Docker File without any extension in my project folder. The script in docker file can be found in code blocks section. This file is a text file that contains a set of instructions used to build my Docker image. Then I downloaded my image via git in my local machine docker by docker build command. After a few minutes I could see my image in my docker desktop.

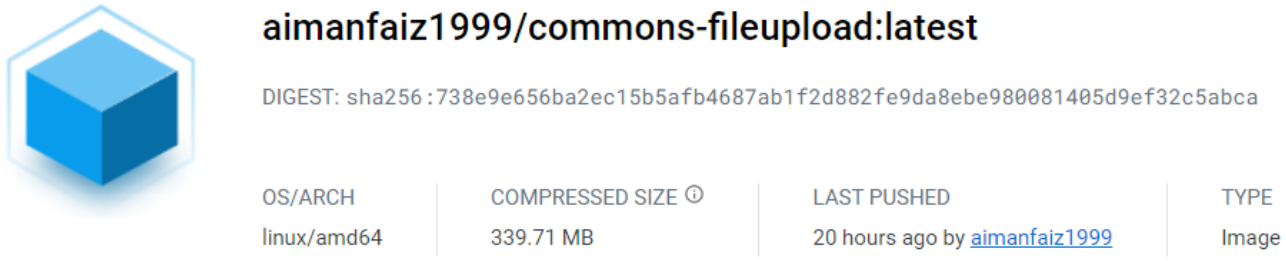
Container running the image:



2.3.2. Publishing image on docker hub

After successfully built of image on my docker desktop, I pushed the image on docker hub by setting up workflow in GitHub actions. The script of the workflow can be found in code

blocks. For setting up the workflow I defined two tokens named "DOCKER_USERNAME" and "DOCKER_PASSWORD" so these keys can authenticate GitHub to my docker hub account. By committing the .yml file my workflow was set up and the image was successfully published on my docker hub account.



2.4. Code coverage

Code coverage measures how much of a software's source code is tested by a specific set of test cases. I have carried out coverage of this project by two ways:

2.4.1. Via JaCoCo

JaCoCo is a code coverage tool designed for Java applications. It provides detailed information about how much Java code is executed during test runs. I integrated JaCoCo into my build process and generated a comprehensive coverage report.

In my report I observed that nearly 18% of instructions are not covered by a test suite. Out of 36 classes 7 classes and out of 226 methods 60 methods were missed respectively. Although the test suite is good enough as it covers more than 80% of code.

Detailed report has been committed to aimanfaiz1999/commons-fileupload repository.

2.4.2. Via Cobertura

It is another popular code coverage tool which provides coverage analysis for Java applications. But for this I had to explicitly define the cobertura plugin in my pom.xml. It is an alternate way of computing code coverage.

2.5. Mutation testing by PI test

Mutation testing is a technique which is used to assess the quality of test suite by introducing fake defects, known as mutations, into the code. Then it checks if tests can detect those mutations or not.

PIT (PITest) is a popular mutation testing framework for Java. PIT stands for "Mutation Testing with JUnit and TestNG."

I started this testing by adding PIT Plugin and dependency with Junit 5 in pom.xml. The script of dependency can be found in the code blocks chapter. After the addition I built the system and ran mvn org.pitest:pitest-maven:mutationCoverage command and it generated two things: 1. Mutations in my code 2. Report. I found Mutation testing report in target/pit-report. Detailed report and all the mutations can be found in pit-reports folder at aimanfaiz1999/commons-fileupload repository.

2.5.1 Results and findings:

This is the comprehensive view of report:

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
26	82%	851/1042	72%	415/575	84%	415/497

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
org.apache.commons.fileupload2	6	87%	418/482	74%	211/284	83%	211/255
org.apache.commons.fileupload2.disk	2	77%	111/145	69%	49/71	84%	49/58
org.apache.commons.fileupload2.impl	2	82%	115/140	65%	45/69	73%	45/62
org.apache.commons.fileupload2.jaksrvlt	3	40%	10/25	31%	4/13	80%	4/5
org.apache.commons.fileupload2.portlet	2	64%	9/14	38%	3/8	75%	3/4
org.apache.commons.fileupload2.pub	3	50%	9/18	20%	1/5	100%	1/1
org.apache.commons.fileupload2.servlet	3	52%	13/25	38%	5/13	83%	5/6
org.apache.commons.fileupload2.util	2	85%	29/34	68%	15/22	83%	15/18
org.apache.commons.fileupload2.util.mime	3	86%	137/159	91%	82/90	93%	82/88

For more on general side, as per the summary, through PI test I mutated my whole project. We mutated 567 lines of code and our test case was only able to kill 416 mutations. 151 lines were not killed by test cases these lines were either ignored or were not covered.

78 lines were not covered by test cases and 82 lines survived that means test cases ran on these lines but were unable to detect the mutation and survived.

The detailed report on GitHub has information about every package and their respective classes.

2.6. *SonarQube analysis for analyzing energy greediness via EcoCode.*

SonarQube is a tool like SonarCloud as it is also used for discovering bugs, code smells and vulnerabilities. But in my project, I am specifically using SonarQube for discovering code smells related to energy which were not discovered during SonarCloud analysis.

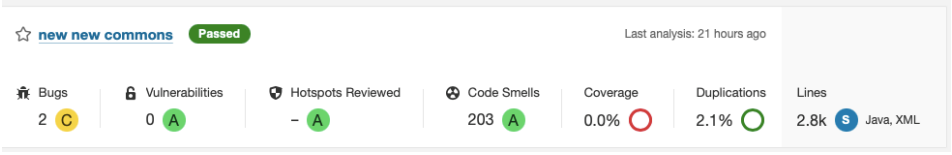
I setup my count of localhost:9000 via docker image. I initialized the project and generated token for my project. Once my project was on SonarCloud, I downloaded EcoCode plugin from marketplace and imported EcoCode rules into my project. After importing the rules, I was able to see 169 code smells related to energy. Other codesmells were the same as SonarCloud.

One of the key codesmell I found was way of incrementing the variable. This analysis suggested using ++I instead of i++.

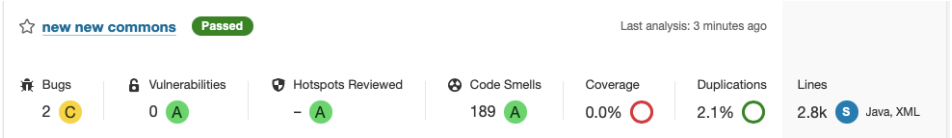
2.6.1. Result and findings

Most code smells were related to Global variable usage in multiple classes and about using switch statements instead of If-else statements.

Before the refactoring:



After Refactoring:



Skipped Refactoring:

1. I skipped removing the unused variables in a particular file. Because removing them was breaking the code.
2. I skipped changing the scope of global variables to local variables because they were supposedly to be used in different clocks of the same file.
3. I was able to refactor the variables scope accurately, but I did not refactor the conditional blocks because they were time taking and required sufficient knowledge.

2.7. Performance testing by Java Microbenchmark Harness

The Java Microbenchmark Harness (JMH) is a framework offered by OpenJDK that allows to create and execute microbenchmarks for Java code. Microbenchmarks are designed to assess the performance and behavior of specific code snippets or methods.

I used JMH for measuring performance of my methods. I started this analysis by adding JMH dependency in my pom.xml and the script can be found in code blocks chapter. I chose setLowerCaseNames() method from parameter parser class. Then I created a benchmark in a new public class using @benchmark annotation.

I imported the relevant packages and annotations from the `org.openjdk.jmh.annotations` package, which is used for benchmarking purposes. My benchmark code sets up a benchmark environment using JMH annotations to measure the performance and average time of the `setLowerCaseNames () ` method in the `ParameterParser` class. The benchmark is configured to run with different values of the `b` parameter, and the `setUp()` method initializes the necessary objects before each benchmark invocation. The benchmark method itself calls the `setLowerCaseNames () ` method to be measured.

Following is the Benchmark Configuration:

@Fork (value = 1, warmups = 2): Specified that benchmark should be forked once and includes warmup iterations. In my case, there are 2 warmup iterations.

@Warmup (iterations = 1): Defined the number of warmup iterations to be performed before the actual measurement iterations. In this case, there is 1 warmup iteration.

@Measurement (iterations = 2): Specifies the number of measurement iterations to be performed. In this case, there are 2 measurement iterations.

@BenchmarkMode (Mode.All): Sets the benchmark mode to measure all available metrics.

2.7.1. Result and findings

Benchmark	(b)	Mode	Cnt	Score	Error	Units
ParameterParserPerformance.PPP.aiman	true	thrpt	2	39497679.453		ops/s
ParameterParserPerformance.PPP.aiman	false	thrpt	2	39358735.683		ops/s
ParameterParserPerformance.PPP.aiman	true	avgt	2	≈ 10^{-8}		s/op
ParameterParserPerformance.PPP.aiman	false	avgt	2	≈ 10^{-8}		s/op
ParameterParserPerformance.PPP.aiman	true	sample	455997	≈ 10^{-8}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.00	true	sample		≈ 0		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.50	true	sample		≈ 0		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.90	true	sample		≈ 10^{-7}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.95	true	sample		≈ 10^{-7}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.99	true	sample		≈ 10^{-7}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.999	true	sample		≈ 10^{-7}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.9999	true	sample		≈ 10^{-6}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p1.00	true	sample		≈ 10^{-4}		s/op
ParameterParserPerformance.PPP.aiman	false	sample	454167	≈ 10^{-8}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.00	false	sample		≈ 0		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.50	false	sample		≈ 0		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.90	false	sample		≈ 10^{-7}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.95	false	sample		≈ 10^{-7}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.99	false	sample		≈ 10^{-7}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.999	false	sample		≈ 10^{-7}		s/op
ParameterParserPerformance.PPP.aiman:aiman·p0.9999	false	sample		≈ 10^{-5}		s/op

2.8. Test cases generation via EvoSuite and Randoop

2.8.1. Test cases generation Via Evosuite

Evosuite is a tool used for automated software testing. It generates unit tests for Java applications. It uses a technique called "search-based software engineering" to automatically generate test cases which results in high code coverage.

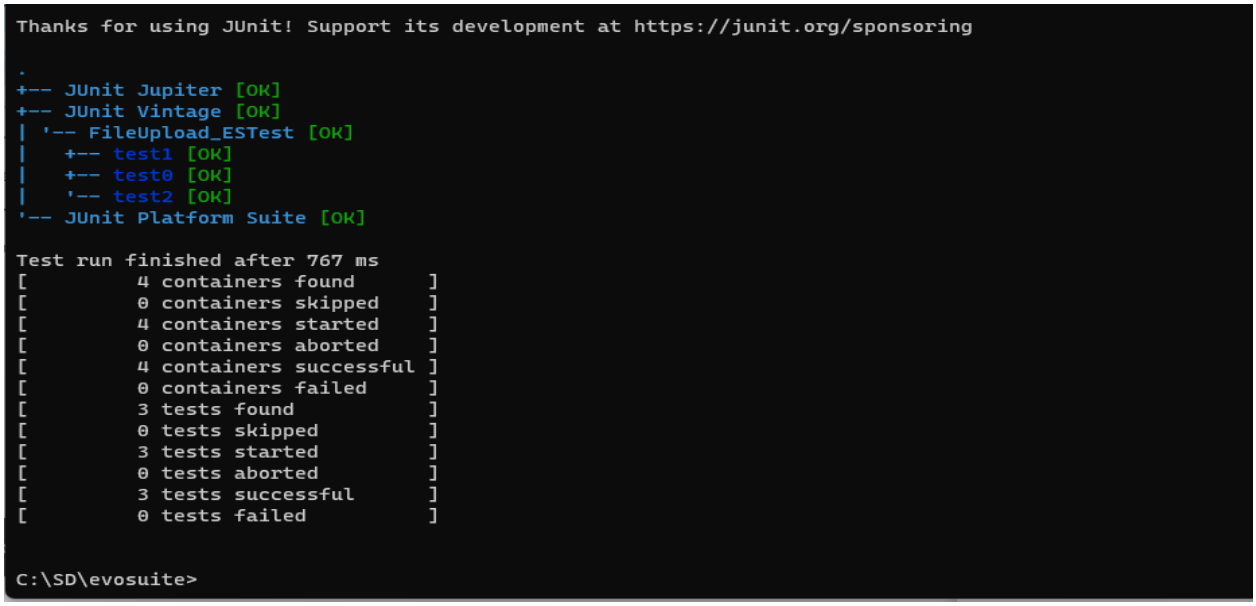
Installation: For making use of evosuite I firstly cloned evosuite on my system. After cloning I added an evosuite-1.2.0.jar file manually in evosuite folder as per the requirement.

Generating tests: After successfully installing, for running the evosuite I choose one class for which I want to generate tests and I choose fileupload.java. In my command, I specified evosuite-1.2.0.jar path, class FQN and project path. In result, the generated tests were stores under /evosuite-test/.

Compilation: But these tests could not be used without compilation. EvoSuite tests require JUnit 4+ and Hamcrest 1.3+. I manually download them from the Maven Central Repository (e.g., JUnit 5.9.1, Hamcrest 2.2).

I compiled the tests using appropriate paths and as a result I got .class files in the same folder as the java file.

Run: Then I ran the generated test cases on my cmd, and I got the following result.



As a result, there was one statistics.csv generated which gives the summary of results. For my class 25 goals, out of 26 goals were covered and the coverage was 93.75%.

All generated test cases and statistic.csv can be found under evosuite-tests and evosuite-report folders on my GitHub repository named aimanfaiz1999/commons-upload.

2.8.2. Regression test cases generation via Randoop

Randoop is another tool used for automated software testing. It generates tests automatically by analyzing the code under the test. Randoop is mainly in unit testing. Developers used this tool to create test suites to validate the functionality of their code.

Installation: For making use of randoop I firstly cloned randoop on my system. After cloning I moved the random-all-4.3.2.jar manually in my target/classes folder as per the requirement. since the Randoop JAR is not executable, so I must explicitly add the JAR in the class path and call the fully qualified name of the class which contain the main () method.

Generating tests: Running randoop includes a prerequisite which is Junit platform console standalone 1.9.2 jar which can be found on maven repository. for generating tests, I choose one class for which I want to test, and I choose fileupload.java. In my command, I specified evosuite-1.2.0.jar path, class FQN and project path. In result, the generated tests were stores under /EvoSuite-test/.

Note: while choosing the class I had to make sure that, the target class is instantiable, i.e., means has at least one accessible constructor

After Randoop ends, the generated tests are stored in the current working directory under randoop-tests/. Generally, Randoop generates two type of test suites:

1. Regression.java
2. Error.java

But in my result, I only got regression test and errors test are optional.

Compilation: But these tests could not be used without compilation. Randoop tests require JUnit 4+ and Hamcrest 1.3+. I manually download them from the Maven Central Repository (e.g., JUnit 5.9.1, Hamcrest 2.2).

I compiled the tests using appropriate paths and as a result I got .class files in the same folder as the java file.

Run: Then I ran the generated test cases on my cmd, and I got the following result.

```
C:\commons-fileupload\target\classes>java -cp randoop-tests;C:\commons-fileupload\target\classes;C:\commons-fileupload\t
arget\classes\randoop-all-4.3.2.jar;C:\randoop\junit-platform-console-standalone-1.9.2.jar org.junit.platform.console.Co
nsoleLauncher -c org.apache.commons.fileupload2.FileItem

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

.
+-- JUnit Jupiter [OK]
+-- JUnit Vintage [OK]
'-- JUnit Platform Suite [OK]

Test run finished after 61 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         0 tests found           ]
[         0 tests skipped         ]
[         0 tests started         ]
[         0 tests aborted         ]
[         0 tests successful      ]
[         0 tests failed          ]


C:\commons-fileupload\target\classes>
```

All generated test cases and statistic.csv can be found under randoop-tests folder on my GitHub repository named aimanfaiz1999/commons-upload.

2.9 Security Analysis with FindSecbugs

FindSecBugs is an analysis which discovers bugs and vulnerabilities related to security. I started this analysis by cloning it locally on my machine. Then I executed findsecbugs.bat which executed successfully.

Note: I used .bat file as I ran it in windows operation system.

I installed FinSecBugs.jar from GitHub. After successful installation of FindSecBugs I ran the .bat file with my project path and as a result I got nothing as there were no potential security bugs in my project. Following attachment shows the result:

```
C:\Users\mtaim\Downloads\findsecbugs-cli-1.12.0>.\findsecbugs.bat C:\SD\commons-fileupload
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by edu.umd.cs.findbugs.ba.jsr305.TypeQualifierValue (file:/C:/Users/
mtaim/Downloads/findsecbugs-cli-1.12.0/lib/spotbugs-4.6.0.jar)
WARNING: Please consider reporting this to the maintainers of edu.umd.cs.findbugs.ba.jsr305.TypeQualifierValue
WARNING: System::setSecurityManager will be removed in a future release

C:\Users\mtaim\Downloads\findsecbugs-cli-1.12.0>
```

2.10. Security Analysis with OWASP DC

This process involves examining the dependencies and matching them with a database of recognized vulnerabilities. After scanning of the project's libraries and dependencies, a detailed report is generated, flagging any detected security concerns or vulnerable components.

I installed OWASP from GitHub. Then ran it with my project directory. It has a cold start phase as it downloads CVE data in directory.

After the complete execution, I got the report in current working directory i.e., release/bin.

I inspected the report, and I found the list of affected dependencies. Additionally, not only relevant dependencies but also test dependencies (e.g., Junit, Hamcrest) and JARs have been analyzed. Detailed report can be found under dependency-check named dependency-check-report All folder on my GitHub repository.

As I am not interested in test dependencies, so I excluded them in my command, and I got the relevant analysis. This report can also be found under the dependency-check folder named dependency-check-report on my GitHub repository.

2.11. Security Analysis with OWASP ZAP

OWASP ZAP (Zed Attack Proxy) is a web application security testing tool. It is used for security assessments and penetration testing of web applications. It is designed for web applications specifically and therefore this analysis is not suitable for my project as it is a java library.

Chapter 3: Code blocks

2.3.1 Docker file Script:

```
# License…..
# limitations under the License.
# syntax=docker/dockerfile:1
FROM maven:3.9.0-eclipse-temurin-17
WORKDIR /app
COPY . .
RUN mvn package
```

2.3.2. Docker publish image workflow:

```
name: Docker image publish
on:
  push:
    branches: [ master ]
jobs:
  docker:
    runs-on: ubuntu-latest
    steps:
      -
        name: Set up QEMU
        uses: docker/setup-qemu-action@v2
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      -
        name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}
      -
        name: Build and push
        uses: docker/build-push-action@v4
```

```
        with:
          push: true
          tags: aimanfaiz1999/commons-fileupload:latest
```

2.5. PIT and Junit dependency:

```
<plugin>
        <groupId>org.pitest</groupId>
        <artifactId>pitest-maven</artifactId>
        <version>1.13.0</version>
        <dependencies>
          <dependency>
            <groupId>org.pitest</groupId>
            <artifactId>pitest-junit5-plugin </artifactId>
            <version>1.1.2</version>
          </dependency>
        </dependencies>
      </plugin>
```

2.7 JMH dependency

```
<dependency>
    <groupId>org.openjdk.jmh</groupId>
    <artifactId>jmh-core</artifactId>
    <version>1.35</version>
</dependency>
<dependency>
    <groupId>org.openjdk.jmh</groupId>
    <artifactId>jmh-generator-annprocess</artifactId>
    <version>1.35</version>
</dependency>
```

2.7 JMH benchmark

```
/**<!--Licensed to the Apache Software Foundation (ASF) under one or more
 contributor license agreements.  See the NOTICE file distributed with
 this work for additional information regarding copyright ownership.
 The ASF licenses this file to You under the Apache License, Version 2.0
```

```
 (the "License"); you may not use this file except in compliance with
 the License.  You may obtain a copy of the License at
 http://www.apache.org/licenses/LICENSE-2.0
 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.--> */
package org.apache.commons.fileupload2.ParameterParserPerformance;
import org.apache.commons.fileupload2.ParameterParser;
import org.openjdk.jmh.annotations.*;
@State(Scope.Benchmark)
public class PPP {
    @Param({ "true", "false" })
    public boolean b;
    ParameterParser pp;
    @Setup(Level.Invocation)
    public void setUp() {
        pp = new ParameterParser();
    }
    @Benchmark
    @Fork(value = 1, warmups = 1)
    @Warmup(iterations = 1)
    @Measurement(iterations = 2)
    @BenchmarkMode(Mode.All)
    public void aiman(PPP plan)
    {
        pp.setLowerCaseNames(plan.b);
    }
}
```

Chapter 4: Conclusion

This report is written to fulfil the semester project requirement of Software dependability course. This report showcases the practical knowledge which I gained in three months during my lessons on this course.

In this report I have mentioned all the software dependability analysis I ran on apache software foundation project name “commons-fileupload” and orchestration of this library on my local docker desktop.

Chapter 5: References

1. Apache Software Foundation. (n.d.). Apache Commons FileUpload. Retrieved from https://projects.apache.org/project.html?commons-fileupload
2. SonarSource. (n.d.). SonarCloud: Continuous Code Quality. Retrieved from https://www.sonarsource.com/products/sonarcloud/
3. GitHub. (n.d.). GitHub. Retrieved from https://github.com/
4. Apache Maven. (n.d.). Maven in Five Minutes. Retrieved from https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
5. Docker Documentation - Dockerizing a Java Application: Docker. (n.d.). Dockerizing a Java Application. Retrieved from https://docs.docker.com/language/java/
6. Educative.io - How do you Dockerize a Maven Project:
7. Educative.io. (n.d.). How do you Dockerize a Maven Project? Retrieved from
8. Baeldung - Cobertura: Baeldung. (n.d.). Cobertura - Introduction and Integration. Retrieved from https://www.baeldung.com/cobertura
9. Codecov Documentation - GitHub Tutorial: Codecov Documentation. (n.d.). GitHub Tutorial. Retrieved from https://docs.codecov.com/docs/github-tutorial
10. JaCoCo - Maven Plugin: JaCoCo. (n.d.). Maven Plugin. Retrieved from https://www.jacoco.org/jacoco/trunk/doc/maven.html
11. Baeldung - Java Mutation Testing with Pitest: Baeldung. (n.d.). Java Mutation Testing with Pitest. Retrieved from https://www.baeldung.com/java-mutation-testing-with-pitest https://www.baeldung.com/java-mutation-testing-with-pitest
12. SonarQube Documentation: SonarSource. (n.d.). SonarQube Documentation. Retrieved from https://docs.sonarqube.org/latest/
13. Green Code Initiative GitHub Repository: Green Code Initiative. (n.d.). Green Code Initiative. Retrieved from https://github.com/green-code-initiative
14. Baeldung. (n.d.). Java Microbenchmark Harness (JMH) - Introduction and Integration. Retrieved from https://www.baeldung.com/java-microbenchmark-harness
15. Evosuite. (n.d.). EvoSuite - Automatic Test Suite Generation for Java. Retrieved from https://www.evosuite.org/
16. emaiannone. (n.d.). tools-tutorial: A tutorial project that covers different development tools. Retrieved from https://github.com/emaiannone/tools-tutorial/tree/master/evosuite
17. Randoop Official Website:
18. Pacheco, C. (n.d.). Randoop: Automated Software Testing for Java. Retrieved from https://randoop.github.io/randoop/
19. emaiannone's GitHub Repository - Randoop: emaiannone. (n.d.). tools-tutorial: A tutorial project that covers different development tools. Retrieved from https://github.com/emaiannone/tools-tutorial/tree/master/randoop
20. FindBugs Security Auditing - FindSecBugs: FindSecBugs. (n.d.). FindBugs Security Auditing - FindSecBugs. Retrieved from https://find-sec-bugs.github.io/
21. emaiannone's GitHub Repository - findsecbugs.bat: emaiannone. (n.d.). tools-tutorial: A tutorial project that covers different development tools. Retrieved from https://github.com/emaiannone/tools-tutorial/blob/master/findsecbugs/findsecbugs.bat
22. OWASP ZAP - Zed Attack Proxy: The OWASP Foundation. (n.d.). OWASP ZAP - Zed Attack Proxy. Retrieved from https://www.zaproxy.org/