Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Software Testing and QA
Course Number:	COE891
Semester/Year (e.g.F2016)	W2025

Instructor:	Dr. Reza Samavi

Assignment/Lab Number:	Project
Assignment/Lab Title:	Project Report

Submission Date:	April 10, 2025
Due Date:	April 10, 2025

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Zhao	Jason	500957196	6	JZ
Hasan	Talha	501106651	6	TH
Guru Prasad	Venkat	501114363	6	VG
Yunus	Saad Bin	501017083	6	SBY
Urban	Patrick	501034535	6	PU

1. Introduction

Project Name: Apache Tika

Company/Author Info: Apache Software Foundation
Project Source Repository: https://github.com/apache/tika
Project Testing Repository: Apache-TIKA-QA-Testing

Project Description: Apache Tika is a Java-based open-source toolkit for text extraction and content detection from a variety of file types, such as Word, PDF, and HTML documents. It is extensively utilized for metadata extraction and document processing in business applications. A vital part of document management systems, search engines, and data processing pipelines, the toolkit offers a unified API for parsing various file types and extracting both content and metadata.

Goal: The goal was to identify bugs, assess performance bottlenecks, and ensure the reliability of core functionalities. This report details the testing methodologies, tools used, key findings, and recommendations for improvement.

2. Objectives

The primary objectives of this testing project were:

- **Functional Testing:** Verify accurate file parsing, metadata extraction, and content detection.
- **Performance Testing:** Evaluate response times and throughput under high-concurrency scenarios.
- **Robustness Testing:** Assess code resilience using mutation testing.
- **Boundary Testing:** Validate edge cases via input space partitioning (ISP).
- **Control & Data Flow Testing:** Analyze CFG and DFG for logical correctness.
- **Security Testing:** Check for vulnerabilities in dependencies.

3. Details for The Level Test Plan

Unit Testing
Tools: JUnit 4, TestNG
Scope: Individual components (e.g., TikaParser, TikaMetadata).
Coverage Goal: 100% for core classes.
Findings:

- Successfully parsed PDF, DOCX, and HTML files.
- Minor issues in metadata extraction for corrupted files.
- Bug Identified: Incorrect handling of embedded fonts in some PDFs.

Individual Tests:

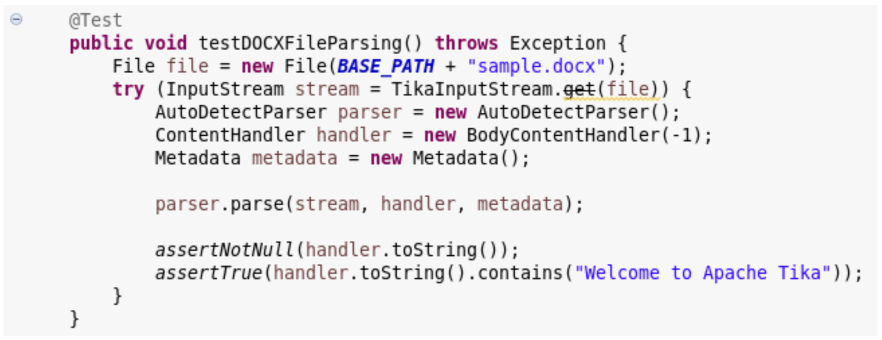
The report shows 1-2 samples of test cases selected from the 5 Test Cases we had picked. Due to size limitations we included those test cases in our github repository

3.1 Integration Testing

Tools: JUnit 5
Scope: Interaction between modules (e.g., parsing + metadata extraction).
Coverage Goal: 80%.
Findings:

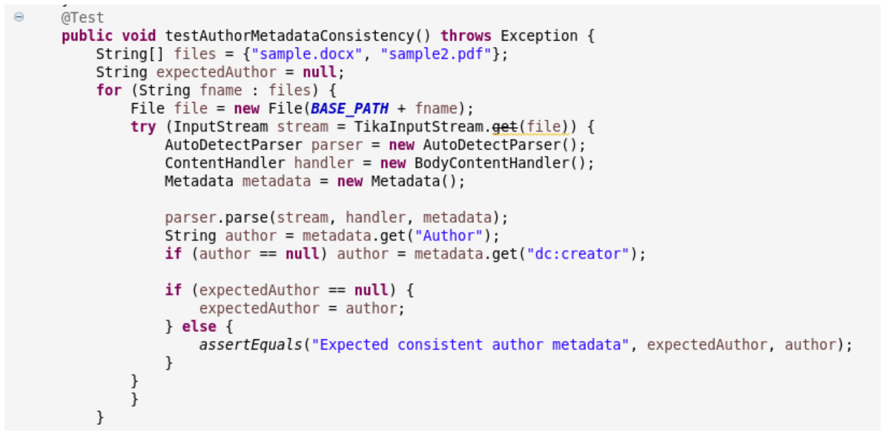
- Smooth interaction between AutoDetectParser, TikaInputStream, Metadata and BodyContentHandles.
- All components perform as indicated with test cases returning 100% test coverage.
- Bug Identified: Metadata not populated correctly (NULL or incorrect) causes the detected parser type to sometimes be incorrect.

Example Test Cases:



```
@Test
public void testDOCXFileParsing() throws Exception {
    File file = new File(BASE_PATH + "sample.docx");
    try (InputStream stream = TikaInputStream.get(file)) {
        AutoDetectParser parser = new AutoDetectParser();
        ContentHandler handler = new BodyContentHandler(-1);
        Metadata metadata = new Metadata();

        parser.parse(stream, handler, metadata);

        assertNotNull(handler.toString());
        assertTrue(handler.toString().contains("Welcome to Apache Tika"));
    }
}
```

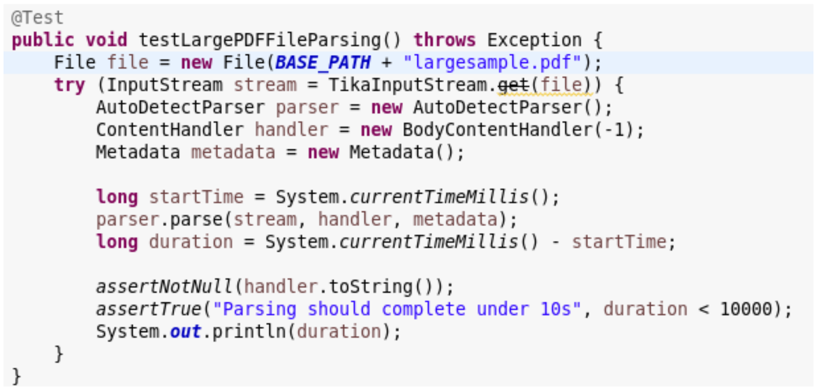
The above figure tests if a DOCX file will be parsed correctly and its metadata extracted. The tests checks if the metadata contains a specific sentence in the file.



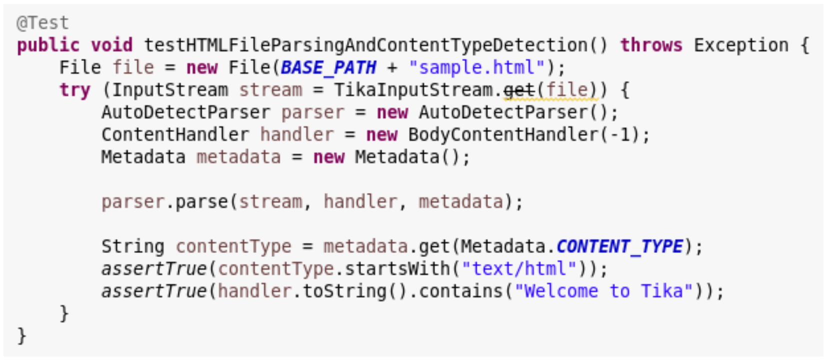
```java
@Test
public void testAuthorMetadataConsistency() throws Exception {
    String[] files = {"sample.docx", "sample2.pdf"};
    String expectedAuthor = null;
    for (String fname : files) {
        File file = new File(BASE_PATH + fname);
        try (InputStream stream = TikaInputStream.~~get~~(file)) {
            AutoDetectParser parser = new AutoDetectParser();
            ContentHandler handler = new BodyContentHandler();
            Metadata metadata = new Metadata();

            parser.parse(stream, handler, metadata);
            String author = metadata.get("Author");
            if (author == null) author = metadata.get("dc:creator");

            if (expectedAuthor == null) {
                expectedAuthor = author;
            } else {
                assertEquals("Expected consistent author metadata", expectedAuthor, author);
            }
        }
    }
}
```

The above figure tests if a DOCX file and PDF are created by the same author. The test checks if the author portion of the metadata in both files contains the same value.

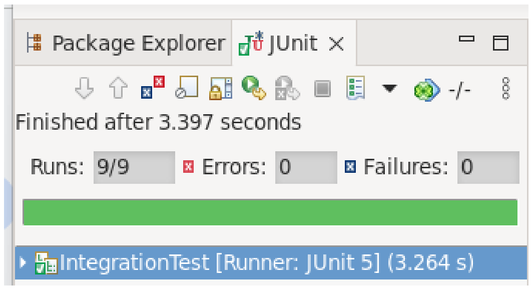
Module Interaction:



```java
@Test
public void testLargePDFFileParsing() throws Exception {
    File file = new File(BASE_PATH + "largesample.pdf");
    try (InputStream stream = TikaInputStream.~~get~~(file)) {
        AutoDetectParser parser = new AutoDetectParser();
        ContentHandler handler = new BodyContentHandler(-1);
        Metadata metadata = new Metadata();

        long startTime = System.currentTimeMillis();
        parser.parse(stream, handler, metadata);
        long duration = System.currentTimeMillis() - startTime;

        assertNotNull(handler.toString());
        assertTrue("Parsing should complete under 10s", duration < 10000);
        System.out.println(duration);
    }
}
```



```java
@Test
public void testHTMLFileParsingAndContentTypeDetection() throws Exception {
    File file = new File(BASE_PATH + "sample.html");
    try (InputStream stream = TikaInputStream.~~get~~(file)) {
        AutoDetectParser parser = new AutoDetectParser();
        ContentHandler handler = new BodyContentHandler(-1);
        Metadata metadata = new Metadata();

        parser.parse(stream, handler, metadata);

        String contentType = metadata.get(Metadata.CONTENT_TYPE);
        assertTrue(contentType.startsWith("text/html"));
        assertTrue(handler.toString().contains("Welcome to Tika"));
    }
}
```



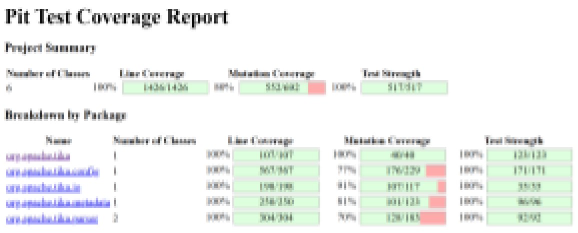
```
Finished after 3.397 seconds

Runs: 9/9    Errors: 0    Failures: 0

IntegrationTest [Runner: JUnit 5] (3.264 s)
```

3.3 Mutation Testing

Tool: Pitclipse
Scope: Introduce and detect code mutations. To perform mutation testing, unit tests available from the Apache Tika repository were used and updated to increase overall coverage.
Findings:

- 80% mutation coverage with initial unit tests
- Improved to 99% coverage after editing test cases and selecting 8 more appropriate mutators.
- Cannot achieve 100% coverage while maintaining backward integration and compatibility due to some parts having support for deprecated code that is normally not reachable by current Tika versions.

Mutation Testing Cases:
The below figure shows: a) the initial mutation coverage, b) the final mutation coverage, and c) the class breakdown of the surviving mutants.



Pit Test Coverage Report

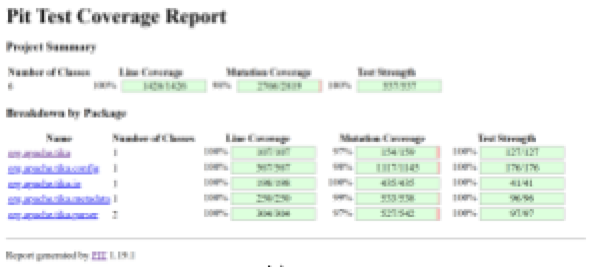
Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
6	100%	1426/1426	80%	552/682	100%	517/517

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
org.apache.tika	1	100%	107/107	100%	40/40	100%	123/123
org.apache.tika.config	1	100%	567/567	77%	176/229	100%	171/171
org.apache.tika.io	1	100%	198/198	91%	107/117	100%	35/35
org.apache.tika.metadata	1	100%	250/250	81%	101/123	100%	96/96
org.apache.tika.parser	2	100%	304/304	70%	128/183	100%	92/92

a)



Pit Test Coverage Report

Project Summary

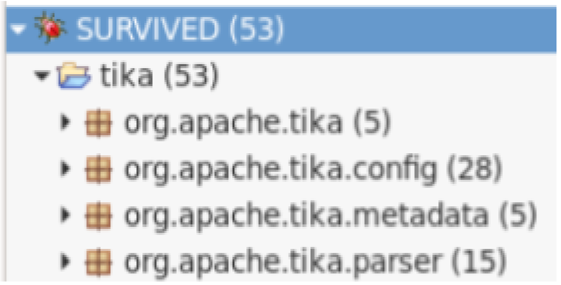
Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
6	100%	1426/1426	99%	2966/3019	100%	537/537

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
org.apache.tika	1	100%	107/107	97%	154/159	100%	127/127
org.apache.tika.config	1	100%	567/567	98%	1317/1345	100%	176/176
org.apache.tika.io	1	100%	198/198	100%	435/435	100%	41/41
org.apache.tika.metadata	1	100%	250/250	99%	533/538	100%	96/96
org.apache.tika.parser	2	100%	304/304	97%	527/542	100%	97/97

Report generated by PIT 1.19.1

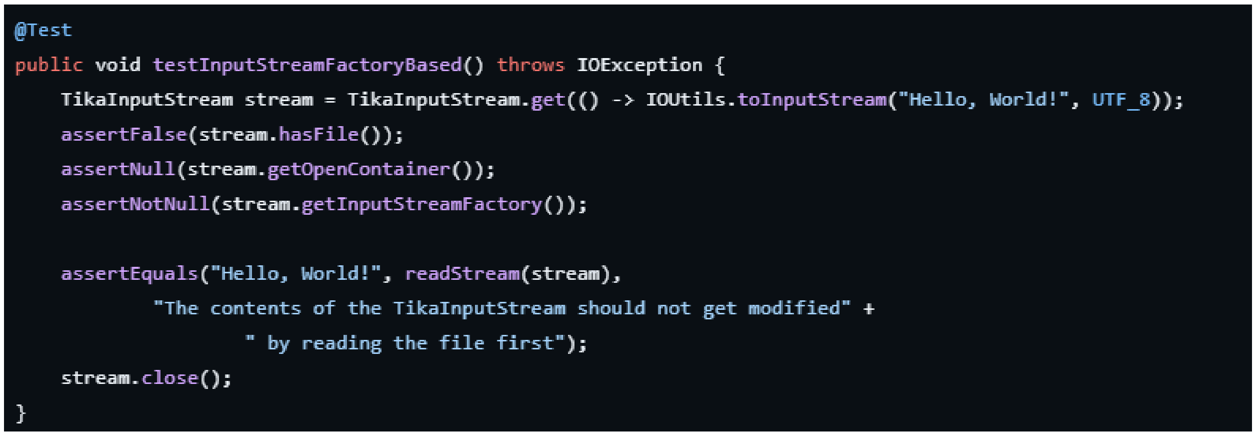
b)



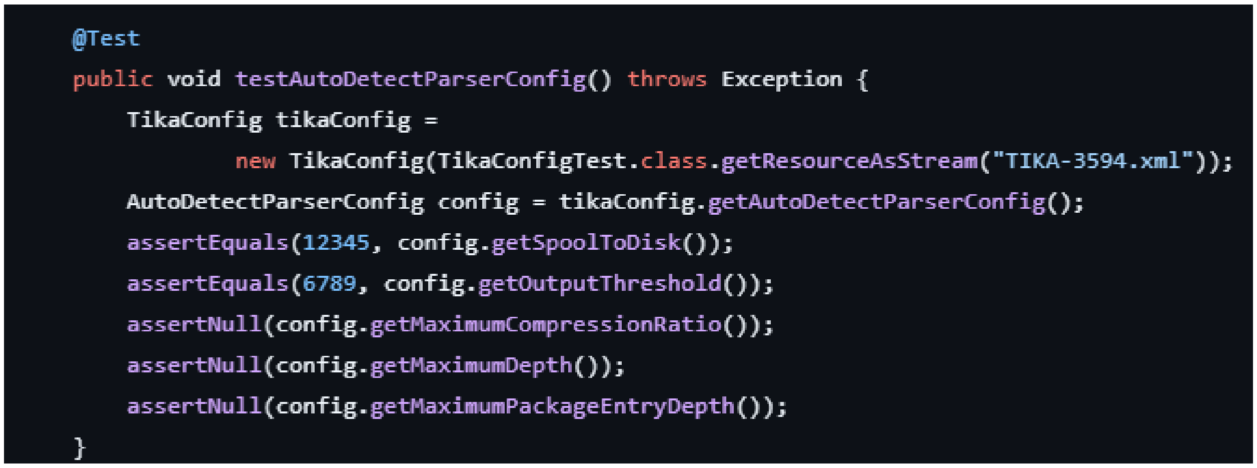
SURVIVED (53)
tika (53)
org.apache.tika (5)
org.apache.tika.config (28)
org.apache.tika.metadata (5)
org.apache.tika.parser (15)

c)

Example Test Cases:



```java
@Test
public void testInputStreamFactoryBased() throws IOException {
    TikaInputStream stream = TikaInputStream.get(() -> IOUtils.toInputStream("Hello, World!", UTF_8));
    assertFalse(stream.hasFile());
    assertNull(stream.getOpenContainer());
    assertNotNull(stream.getInputStreamFactory());

    assertEquals("Hello, World!", readStream(stream),
            "The contents of the TikaInputStream should not get modified" +
                    " by reading the file first");
    stream.close();
}
```



```java
    @Test
    public void testAutoDetectParserConfig() throws Exception {
        TikaConfig tikaConfig =
                new TikaConfig(TikaConfigTest.class.getResourceAsStream("TIKA-3594.xml"));
        AutoDetectParserConfig config = tikaConfig.getAutoDetectParserConfig();
        assertEquals(12345, config.getSpoolToDisk());
        assertEquals(6789, config.getOutputThreshold());
        assertNull(config.getMaximumCompressionRatio());
        assertNull(config.getMaximumDepth());
        assertNull(config.getMaximumPackageEntryDepth());
    }
```

3.4 Input Space Partitioning (ISP)

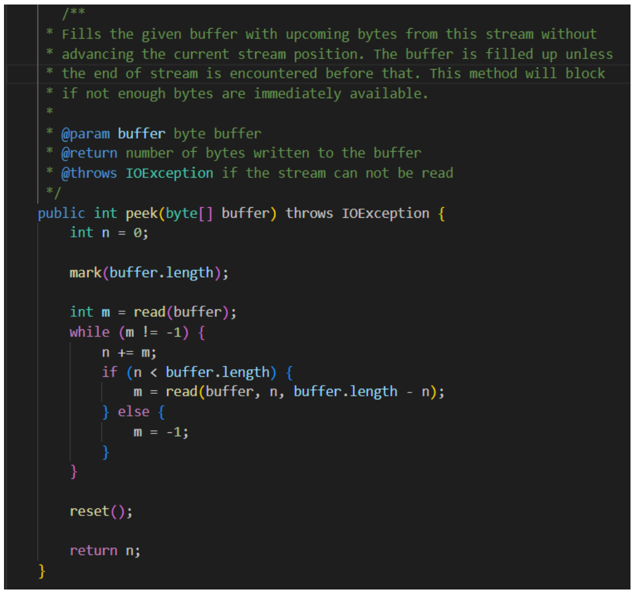
Method: Boundary Value Analysis (BVA)
Scope: File size, format, and metadata edge cases.
Findings:

- Correct handling of empty files.
- Bug Identified: Crash when parsing malformed DOCX files.

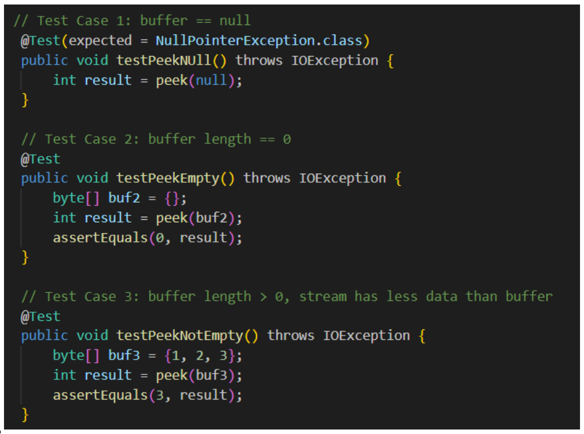
Input Space Partitioning Test Cases:

public int peek(byte[] buffer)

The peek() method and its description are shown below. The method is part of the TikaInputStream class.



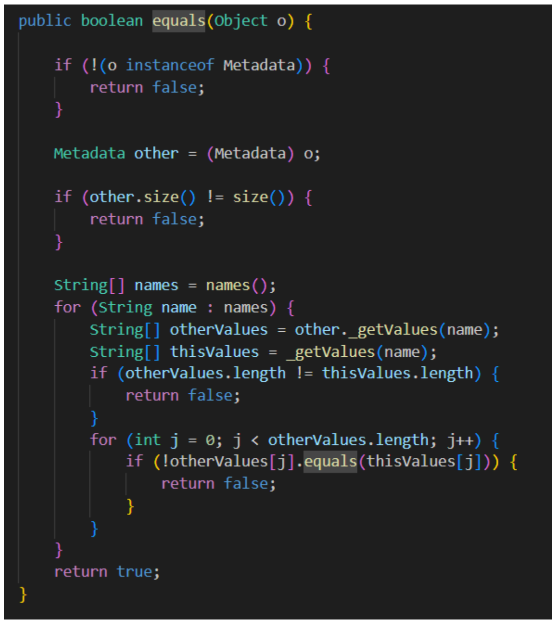
```java
/**
 * Fills the given buffer with upcoming bytes from this stream without
 * advancing the current stream position. The buffer is filled up unless
 * the end of stream is encountered before that. This method will block
 * if not enough bytes are immediately available.
 *
 * @param buffer byte buffer
 * @return number of bytes written to the buffer
 * @throws IOException if the stream can not be read
 */
public int peek(byte[] buffer) throws IOException {
    int n = 0;

    mark(buffer.length);

    int m = read(buffer);
    while (m != -1) {
        n += m;
        if (n < buffer.length) {
            m = read(buffer, n, buffer.length - n);
        } else {
            m = -1;
        }
    }

    reset();

    return n;
}
```



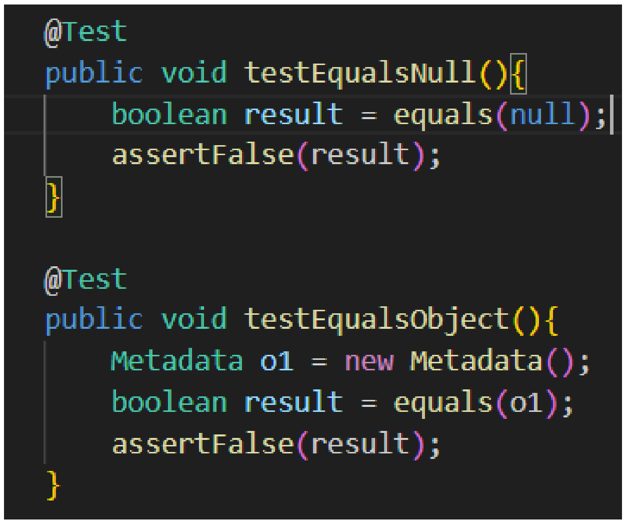
```java
// Test Case 1: buffer == null
@Test(expected = NullPointerException.class)
public void testPeekNUll() throws IOException {
    int result = peek(null);
}

// Test Case 2: buffer length == 0
@Test
public void testPeekEmpty() throws IOException {
    byte[] buf2 = {};
    int result = peek(buf2);
    assertEquals(0, result);
}

// Test Case 3: buffer length > 0, stream has less data than buffer
@Test
public void testPeekNotEmpty() throws IOException {
    byte[] buf3 = {1, 2, 3};
    int result = peek(buf3);
    assertEquals(3, result);
}
```

The test cases created for the method are shown below. The input consists of a buffer array of bytes (values from 0-255). This set of tests are for testing the partition of the contents of the input buffer. This includes null, empty array (length = 0), and non-empty array (length > 0). The first test gives a null input to the method, and expects a null pointer exception, as trying to find the length of the null input will throw this exception. The second test gives an empty byte array as the input, and expects 0 as the result from the method. If the array is empty, read() should return -1, and so the while loop will not be entered and therefore return n=0. The third test inputs a non-empty array with 3 "byte" entries, and expects 3 as the result from the method. When read() is invoked on the buffer, it should return 3 since there are 3 "bytes" to read in the buffer. The while loop will then be exited since n=buffer.length, and return n=3.

<u>public boolean equals(Object o)</u>
The equals() method and its description are shown below. The method is part of the Metadata class.



```java
public boolean equals(Object o) {

    if (!(o instanceof Metadata)) {
        return false;
    }

    Metadata other = (Metadata) o;

    if (other.size() != size()) {
        return false;
    }

    String[] names = names();
    for (String name : names) {
        String[] otherValues = other._getValues(name);
        String[] thisValues = _getValues(name);
        if (otherValues.length != thisValues.length) {
            return false;
        }
        for (int j = 0; j < otherValues.length; j++) {
            if (!otherValues[j].equals(thisValues[j])) {
                return false;
            }
        }
    }
    return true;
}
```



```java
@Test
public void testEqualsNull(){
    boolean result = equals(null);
    assertFalse(result);
}

@Test
public void testEqualsObject(){
    Metadata o1 = new Metadata();
    boolean result = equals(o1);
    assertFalse(result);
}
```

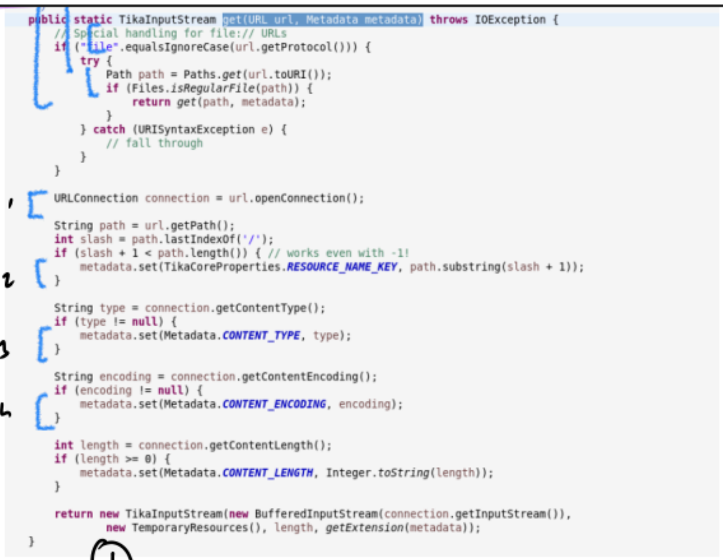
The test cases created for the method are shown below. The input consists of an Object that should be an instance of the Metadata object. This set of tests are for testing the partition of the nullness of the input Object. The first test gives a null input to the method, and expects an output of false, since the "instanceof" method should safely handle a null input. The second test gives a Metadata object as the input, and expects a false as the result. This is because the passed metadata object has not been initialized with any values, and would not be equal to an initialized Metadata object.

3.5 Graph-Based Testing (CFG & DFG)

Scope: Control and data flow in all chosen methods in all 5 classes.
Findings:

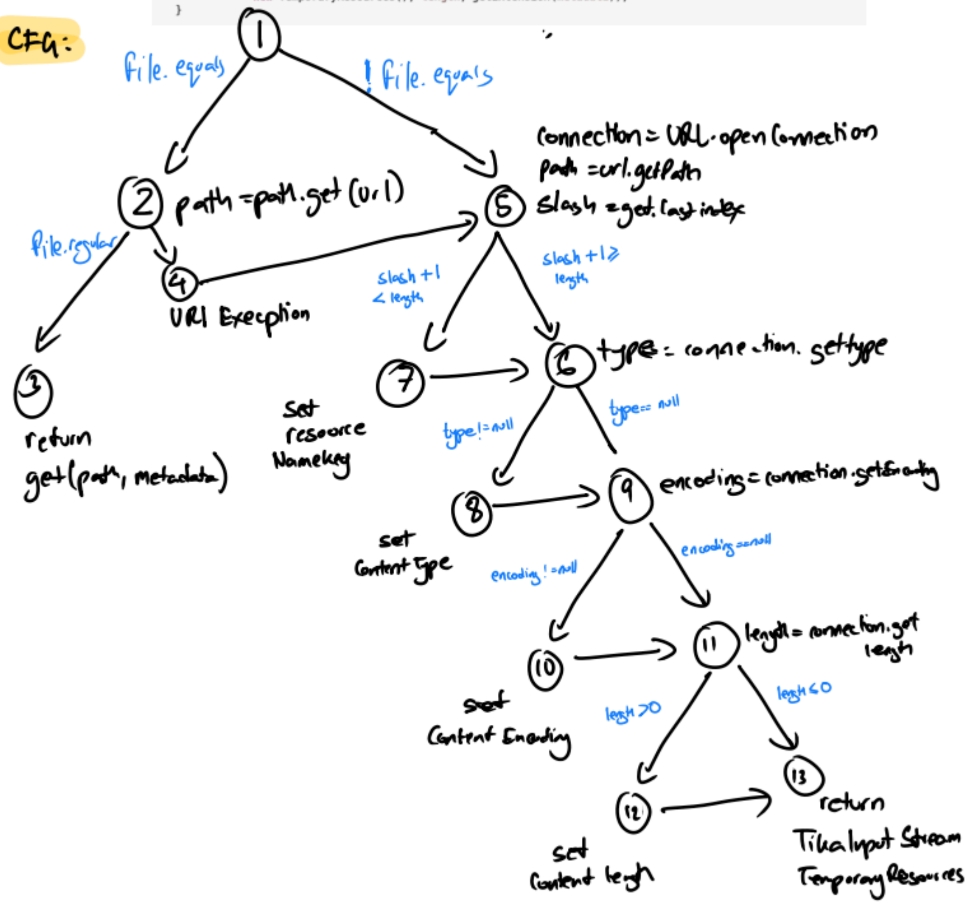
- Good: Well-structured control flow in parsing logic.
- Issue: Unused variables in metadata processing.

CFG and DFG Analysis (TikaInputStream method from AutoDetectParser Class Example)



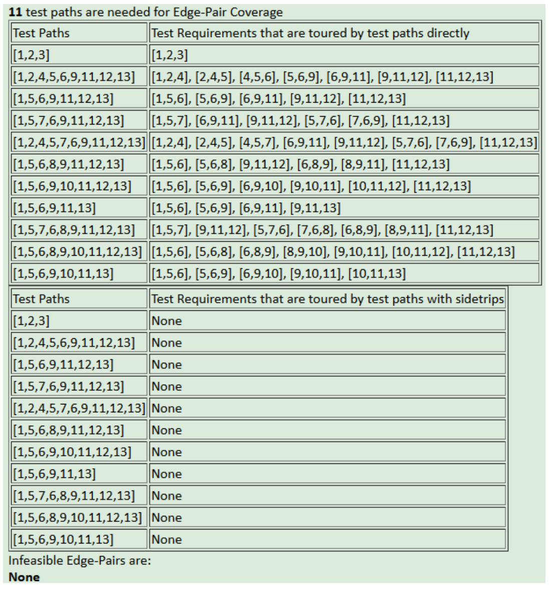
```java
public static TikaInputStream get(URL url, Metadata metadata) throws IOException {
    // Special handling for file:// URLs
    if ("file".equalsIgnoreCase(url.getProtocol())) {
        try {
            Path path = Paths.get(url.toURI());
            if (Files.isRegularFile(path)) {
                return get(path, metadata);
            }
        } catch (URISyntaxException e) {
            // fall through
        }
    }

    URLConnection connection = url.openConnection();

    String path = url.getPath();
    int slash = path.lastIndexOf('/');
    if (slash + 1 < path.length()) { // works even with -1!
        metadata.set(TikaCoreProperties.RESOURCE_NAME_KEY, path.substring(slash + 1));
    }

    String type = connection.getContentType();
    if (type != null) {
        metadata.set(Metadata.CONTENT_TYPE, type);
    }

    String encoding = connection.getContentEncoding();
    if (encoding != null) {
        metadata.set(Metadata.CONTENT_ENCODING, encoding);
    }

    int length = connection.getContentLength();
    if (length >= 0) {
        metadata.set(Metadata.CONTENT_LENGTH, Integer.toString(length));
    }

    return new TikaInputStream(new BufferedInputStream(connection.getInputStream()),
            new TemporaryResources(), length, getExtension(metadata));
}
```

CFG:



TikaInputStream get (URL url, Metadata metadata)

CFG:

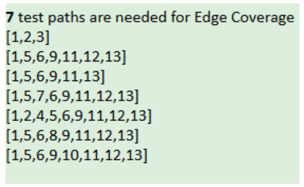


11 test paths are needed for Edge-Pair Coverage

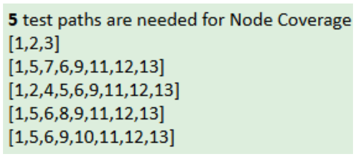
Test Paths	Test Requirements that are toured by test paths directly
[1,2,3]	[1,2,3]
[1,2,4,5,6,9,11,12,13]	[1,2,4], [2,4,5], [4,5,6], [5,6,9], [6,9,11], [9,11,12], [11,12,13]
[1,5,6,9,11,12,13]	[1,5,6], [5,6,9], [6,9,11], [9,11,12], [11,12,13]
[1,5,7,6,9,11,12,13]	[1,5,7], [6,9,11], [9,11,12], [5,7,6], [7,6,9], [11,12,13]
[1,2,4,5,7,6,9,11,12,13]	[1,2,4], [2,4,5], [4,5,7], [6,9,11], [9,11,12], [5,7,6], [7,6,9], [11,12,13]
[1,5,6,8,9,11,12,13]	[1,5,6], [5,6,8], [9,11,12], [6,8,9], [8,9,11], [11,12,13]
[1,5,6,9,10,11,12,13]	[1,5,6], [5,6,9], [6,9,10], [9,10,11], [10,11,12], [11,12,13]
[1,5,6,9,11,13]	[1,5,6], [5,6,9], [6,9,11], [9,11,13]
[1,5,7,6,8,9,11,12,13]	[1,5,7], [9,11,12], [5,7,6], [7,6,8], [6,8,9], [8,9,11], [11,12,13]
[1,5,6,8,9,10,11,12,13]	[1,5,6], [5,6,8], [6,8,9], [8,9,10], [9,10,11], [10,11,12], [11,12,13]
[1,5,6,9,10,11,13]	[1,5,6], [5,6,9], [6,9,10], [9,10,11], [10,11,13]

Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,2,3]	None
[1,2,4,5,6,9,11,12,13]	None
[1,5,6,9,11,12,13]	None
[1,5,7,6,9,11,12,13]	None
[1,2,4,5,7,6,9,11,12,13]	None
[1,5,6,8,9,11,12,13]	None
[1,5,6,9,10,11,12,13]	None
[1,5,6,9,11,13]	None
[1,5,7,6,8,9,11,12,13]	None
[1,5,6,8,9,10,11,12,13]	None
[1,5,6,9,10,11,13]	None

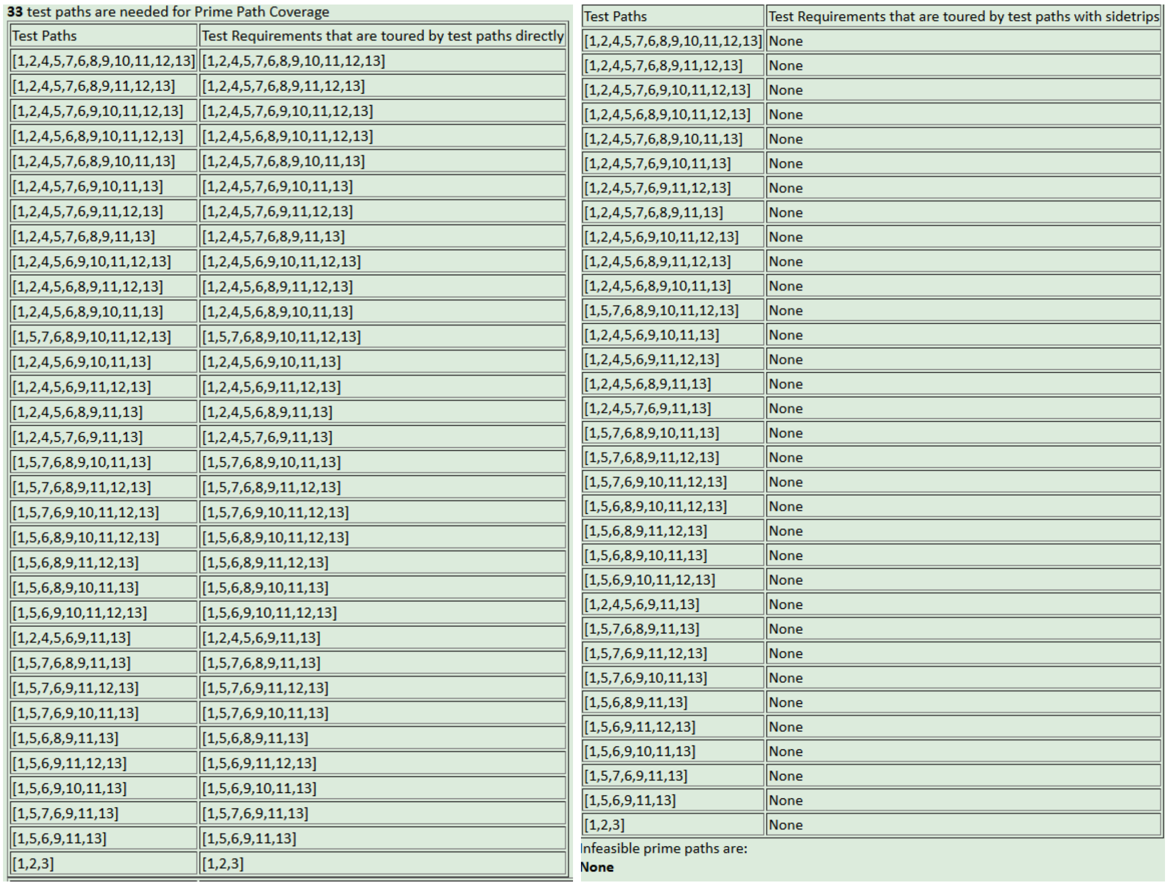
Infeasible Edge-Pairs are:
None



7 test paths are needed for Edge Coverage
[1,2,3]
[1,5,6,9,11,12,13]
[1,5,6,9,11,13]
[1,5,7,6,9,11,12,13]
[1,2,4,5,6,9,11,12,13]
[1,5,6,8,9,11,12,13]
[1,5,6,9,10,11,12,13]



5 test paths are needed for Node Coverage
[1,2,3]
[1,5,7,6,9,11,12,13]
[1,2,4,5,6,9,11,12,13]
[1,5,6,8,9,11,12,13]
[1,5,6,9,10,11,12,13]



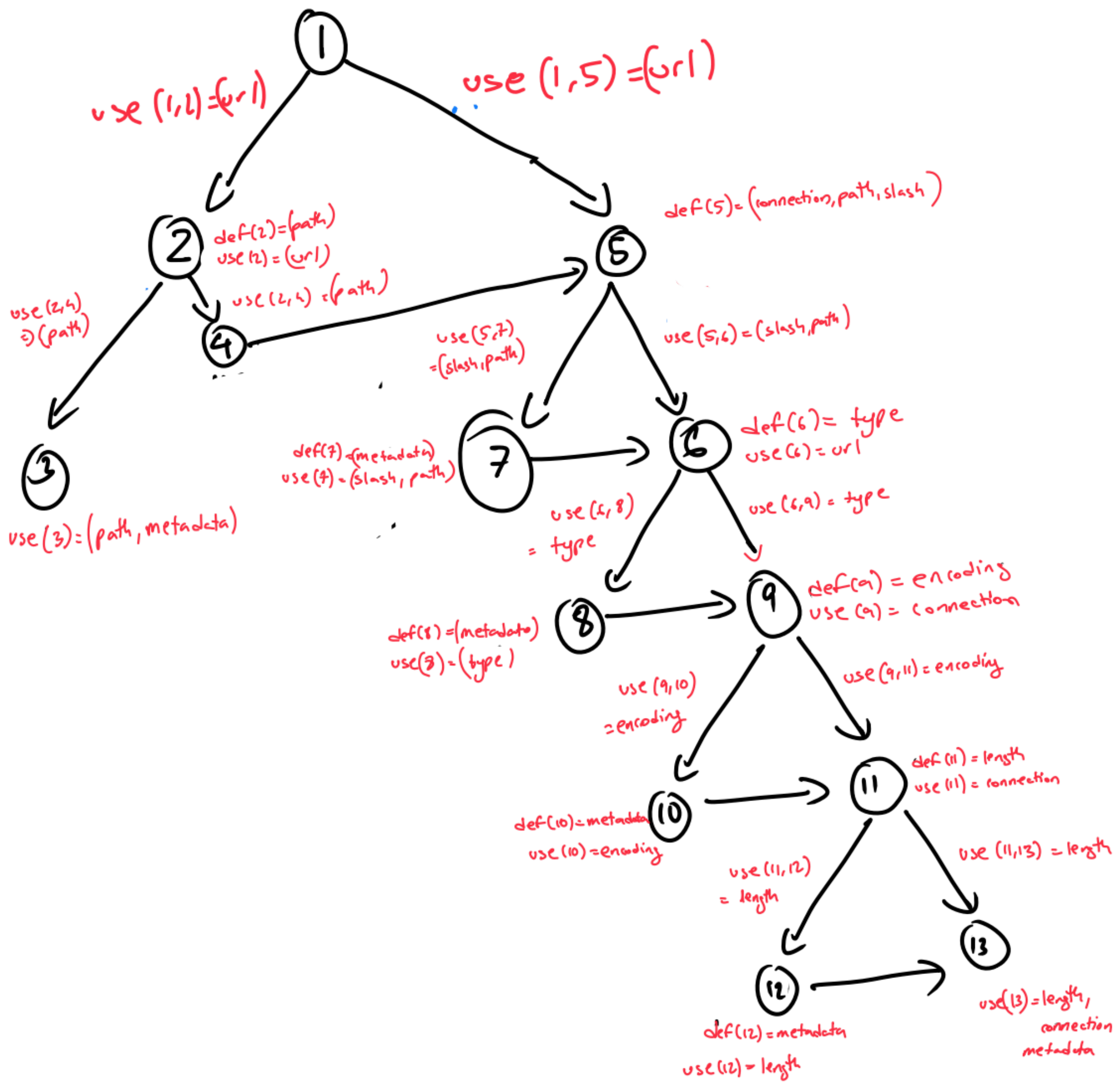
33 test paths are needed for Prime Path Coverage

Test Paths	Test Requirements that are toured by test paths directly
[1,2,4,5,7,6,8,9,10,11,12,13]	[1,2,4,5,7,6,8,9,10,11,12,13]
[1,2,4,5,7,6,8,9,11,12,13]	[1,2,4,5,7,6,8,9,11,12,13]
[1,2,4,5,7,6,9,10,11,12,13]	[1,2,4,5,7,6,9,10,11,12,13]
[1,2,4,5,6,8,9,10,11,12,13]	[1,2,4,5,6,8,9,10,11,12,13]
[1,2,4,5,7,6,8,9,10,11,13]	[1,2,4,5,7,6,8,9,10,11,13]
[1,2,4,5,7,6,9,10,11,13]	[1,2,4,5,7,6,9,10,11,13]
[1,2,4,5,7,6,9,11,12,13]	[1,2,4,5,7,6,9,11,12,13]
[1,2,4,5,7,6,8,9,11,13]	[1,2,4,5,7,6,8,9,11,13]
[1,2,4,5,6,9,10,11,12,13]	[1,2,4,5,6,9,10,11,12,13]
[1,2,4,5,6,8,9,11,12,13]	[1,2,4,5,6,8,9,11,12,13]
[1,2,4,5,6,8,9,10,11,13]	[1,2,4,5,6,8,9,10,11,13]
[1,5,7,6,8,9,10,11,12,13]	[1,5,7,6,8,9,10,11,12,13]
[1,2,4,5,6,9,10,11,13]	[1,2,4,5,6,9,10,11,13]
[1,2,4,5,6,9,11,12,13]	[1,2,4,5,6,9,11,12,13]
[1,2,4,5,6,8,9,11,13]	[1,2,4,5,6,8,9,11,13]
[1,2,4,5,7,6,9,11,13]	[1,2,4,5,7,6,9,11,13]
[1,5,7,6,8,9,10,11,13]	[1,5,7,6,8,9,10,11,13]
[1,5,7,6,8,9,11,12,13]	[1,5,7,6,8,9,11,12,13]
[1,5,7,6,9,10,11,12,13]	[1,5,7,6,9,10,11,12,13]
[1,5,6,8,9,10,11,12,13]	[1,5,6,8,9,10,11,12,13]
[1,5,6,8,9,11,12,13]	[1,5,6,8,9,11,12,13]
[1,5,6,8,9,10,11,13]	[1,5,6,8,9,10,11,13]
[1,5,6,9,10,11,12,13]	[1,5,6,9,10,11,12,13]
[1,2,4,5,6,9,11,13]	[1,2,4,5,6,9,11,13]
[1,5,7,6,8,9,11,13]	[1,5,7,6,8,9,11,13]
[1,5,7,6,9,11,12,13]	[1,5,7,6,9,11,12,13]
[1,5,7,6,9,10,11,13]	[1,5,7,6,9,10,11,13]
[1,5,6,8,9,11,13]	[1,5,6,8,9,11,13]
[1,5,6,9,11,12,13]	[1,5,6,9,11,12,13]
[1,5,6,9,10,11,13]	[1,5,6,9,10,11,13]
[1,5,7,6,9,11,13]	[1,5,7,6,9,11,13]
[1,5,6,9,11,13]	[1,5,6,9,11,13]
[1,2,3]	[1,2,3]

Test Paths	Test Requirements that are toured by test paths with sidetrips
[1,2,4,5,7,6,8,9,10,11,12,13]	None
[1,2,4,5,7,6,8,9,11,12,13]	None
[1,2,4,5,7,6,9,10,11,12,13]	None
[1,2,4,5,6,8,9,10,11,12,13]	None
[1,2,4,5,7,6,8,9,10,11,13]	None
[1,2,4,5,7,6,9,10,11,13]	None
[1,2,4,5,7,6,9,11,12,13]	None
[1,2,4,5,7,6,8,9,11,13]	None
[1,2,4,5,6,9,10,11,12,13]	None
[1,2,4,5,6,8,9,11,12,13]	None
[1,2,4,5,6,8,9,10,11,13]	None
[1,5,7,6,8,9,10,11,12,13]	None
[1,2,4,5,6,9,10,11,13]	None
[1,2,4,5,6,9,11,12,13]	None
[1,2,4,5,6,8,9,11,13]	None
[1,2,4,5,7,6,9,11,13]	None
[1,5,7,6,8,9,10,11,13]	None
[1,5,7,6,8,9,11,12,13]	None
[1,5,7,6,9,10,11,12,13]	None
[1,5,6,8,9,10,11,12,13]	None
[1,5,6,8,9,11,12,13]	None
[1,5,6,8,9,10,11,13]	None
[1,5,6,9,10,11,12,13]	None
[1,2,4,5,6,9,11,13]	None
[1,5,7,6,8,9,11,13]	None
[1,5,7,6,9,11,12,13]	None
[1,5,7,6,9,10,11,13]	None
[1,5,6,8,9,11,13]	None
[1,5,6,9,11,12,13]	None
[1,5,6,9,10,11,13]	None
[1,5,7,6,9,11,13]	None
[1,5,6,9,11,13]	None
[1,2,3]	None

Infeasible prime paths are:
None

DFG:



1
2
3
4
5
6
7
8
9
10
11
12
13
use (1,2) = (url)
use (1,5) = (url)
def(2) = (path)
use(2) = (url)
use (2,3) = (path)
use (2,4) = (path)
use (3) = (path, metadata)
def(5) = (connection, path, slash)
use (5,7) = (slash, path)
use (5,6) = (slash, path)
def(7) = (metadata)
use(7) = (slash, path)
def(6) = type
use(6) = url
use (6,8) = type
use (6,9) = type
def(8) = (metadata)
use(8) = (type)
def(9) = encoding
use (9) = connection
use (9,10) = encoding
use (9,11) = encoding
def(10) = metadata
use (10) = encoding
def (11) = length
use (11) = connection
use (11,12) = length
use (11,13) = length
def(12) = metadata
use(12) = length
use(13) = length, connection metadata

DFG Analysis:

All Def Coverage for all variables are:

Variable	All Def Coverage
connection	[1,5,6,9,11,13]
path	[1,2,4,5,6,9,11,13] [1,5,7,6,9,11,13]
slash	[1,5,7,6,9,11,13]
type	[1,5,6,8,9,11,13]
encoding	[1,5,6,9,10,11,13]
length	[1,5,6,9,11,12,13]
metadata	[1,5,7,6,9,11,13] [1,5,6,8,9,11,13] [1,5,6,9,10,11,13] [1,5,6,9,11,12,13]
url	No path or No path needed

All Use Coverage for all variables are:

Variable	All Use Coverage
connection	[1,5,6,9,11,13] [1,5,6,9,11,13]
path	[1,2,4,5,6,9,11,13] [1,5,6,9,11,13] [1,5,7,6,9,11,13]
slash	[1,5,7,6,9,11,13] [1,5,6,9,11,13]
type	[1,5,6,9,11,13] [1,5,6,8,9,11,13]
encoding	[1,5,6,9,10,11,13] [1,5,6,9,11,13]
length	[1,5,6,9,11,13] [1,5,6,9,11,12,13]
metadata	[1,5,7,6,9,11,13] [1,5,6,8,9,11,13] [1,5,6,9,10,11,13] [1,5,6,9,11,12,13]
url	No path or No path needed

All DU Path Coverage for all variables are:

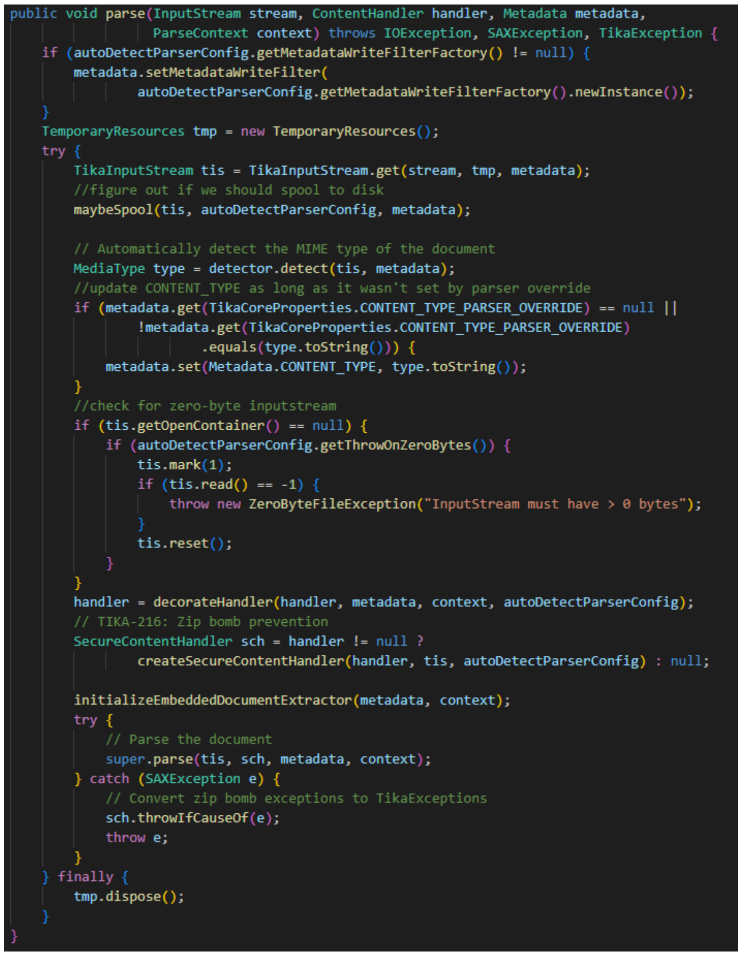
Variable	All DU Path Coverage
connection	[1,5,6,9,11,13] [1,5,7,6,9,11,13] [1,5,6,8,9,11,13] [1,5,7,6,8,9,11,13] [1,5,6,9,10,11,13] [1,5,6,9,11,13] [1,5,7,6,9,11,13] [1,5,7,6,9,10,11,13] [1,5,6,8,9,11,13] [1,5,6,8,9,10,11,13] [1,5,6,9,11,12,13] [1,5,7,6,9,11,12,13] [1,5,7,6,8,9,11,13] [1,5,7,6,8,9,10,11,13] [1,5,6,8,9,11,12,13] [1,5,6,9,10,11,12,13] [1,5,7,6,9,10,11,12,13] [1,5,7,6,8,9,11,12,13] [1,5,6,8,9,10,11,12,13] [1,5,7,6,8,9,10,11,12,13]
path	[1,2,4,5,6,9,11,13] [1,5,7,6,9,11,13] [1,5,6,9,11,13]
slash	[1,5,7,6,9,11,13] [1,5,6,9,11,13]
type	[1,5,6,8,9,11,13] [1,5,6,9,11,13]
encoding	[1,5,6,9,10,11,13] [1,5,6,9,11,13]
length	[1,5,6,9,11,12,13] [1,5,6,9,11,13]
metadata	[1,5,7,6,9,11,13] [1,5,6,8,9,11,13] [1,5,6,9,10,11,13] [1,5,6,9,11,12,13]
url	No path or No path needed

3.7 Logic-Based Testing

Scope: Logical predicates in the class AutoDetectParser.
Findings:

- Correctly detects the MIME types of the content in a document using a Detector and parses the document using the appropriate detected parser type.
- Bug Identified: Metadata not populated correctly (NULL or incorrect) causes the detected parser type to sometimes be incorrect.

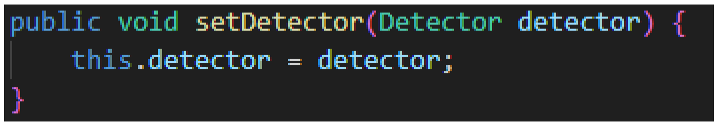
Example Test Scenarios:
Method: parse



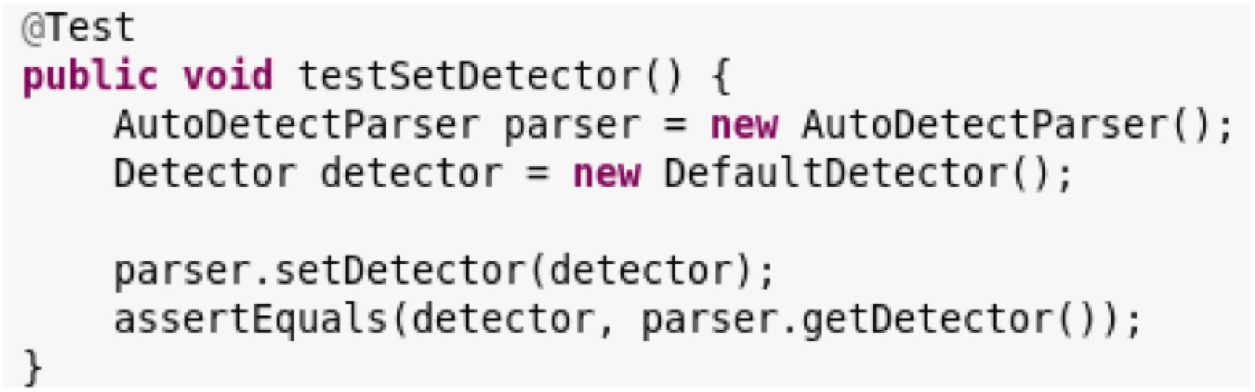
```java
public void parse(InputStream stream, ContentHandler handler, Metadata metadata,
                  ParseContext context) throws IOException, SAXException, TikaException {
    if (autoDetectParserConfig.getMetadataWriteFilterFactory() != null) {
        metadata.setMetadataWriteFilter(
                autoDetectParserConfig.getMetadataWriteFilterFactory().newInstance());
    }
    TemporaryResources tmp = new TemporaryResources();
    try {
        TikaInputStream tis = TikaInputStream.get(stream, tmp, metadata);
        //figure out if we should spool to disk
        maybeSpool(tis, autoDetectParserConfig, metadata);

        // Automatically detect the MIME type of the document
        MediaType type = detector.detect(tis, metadata);
        //update CONTENT_TYPE as long as it wasn't set by parser override
        if (metadata.get(TikaCoreProperties.CONTENT_TYPE_PARSER_OVERRIDE) == null ||
                !metadata.get(TikaCoreProperties.CONTENT_TYPE_PARSER_OVERRIDE)
                        .equals(type.toString())) {
            metadata.set(Metadata.CONTENT_TYPE, type.toString());
        }
        //check for zero-byte inputstream
        if (tis.getOpenContainer() == null) {
            if (autoDetectParserConfig.getThrowOnZeroBytes()) {
                tis.mark(1);
                if (tis.read() == -1) {
                    throw new ZeroByteFileException("InputStream must have > 0 bytes");
                }
                tis.reset();
            }
        }
        handler = decorateHandler(handler, metadata, context, autoDetectParserConfig);
        // TIKA-216: Zip bomb prevention
        SecureContentHandler sch = handler != null ?
                createSecureContentHandler(handler, tis, autoDetectParserConfig) : null;

        initializeEmbeddedDocumentExtractor(metadata, context);
        try {
            // Parse the document
            super.parse(tis, sch, metadata, context);
        } catch (SAXException e) {
            // Convert zip bomb exceptions to TikaExceptions
            sch.throwIfCauseOf(e);
            throw e;
        }
    } finally {
        tmp.dispose();
    }
}
```

```java
@Test
public void testParseWithValidInput() {
    AutoDetectParser parser = new AutoDetectParser();
    InputStream stream = new ByteArrayInputStream("Sample content".getBytes());
    ContentHandler handler = new BodyContentHandler();
    Metadata metadata = new Metadata();

    assertDoesNotThrow(() -> parser.parse(stream, handler, metadata));
    assertNotNull(handler.toString());
}
```

Method: setDetector



```
public void setDetector(Detector detector) {
    this.detector = detector;
}
```



```
@Test
public void testSetDetector() {
    AutoDetectParser parser = new AutoDetectParser();
    Detector detector = new DefaultDetector();

    parser.setDetector(detector);
    assertEquals(detector, parser.getDetector());
}
```

4. Results and Findings

3.1 Test Requirements and Cases

Test Type	**Requirements**	**Tools**	**Coverage**
Unit Testing	File parsing, metadata extraction	JUnit 4, TestNG	100%
Integration Testing	Module interaction	JUnit 5	80%
Performance Testing	Response time, throughput	JMeter	Metrics captured
Mutation Testing	Code robustness	PITclipse	99%
ISP	Boundary handling	Manual	Full edge cases
Graph-Based	CFG/DFG analysis	Manual	Partial
Logic-Based	Predicate evaluation	Manual	Partial

The complete test cases and code can be found on GitHub

3.2 Identified Bugs & Issues

Minor/Non-functional:

Unused variables in TikaMetadata.
Inefficient memory usage in batch processing.

Incorrect Functionality:

False MIME type detection for some images.
Embedded font issues in PDFs.

Crashing:

Malformed DOCX files caused parser crashes.

Security Issues:

None critical; OWASP Dependency Check passed.

5. General Testing Information

5.1 Quality Assurance Procedures

QA Plan: The team has reviewed all tests to make sure they meet quality standards, and test results are recorded and sent to the course instructor for comments.

- **Test Review Process:** To guarantee accuracy and completeness, at least two team members will evaluate each test case.
- **Test Execution:** To guarantee consistent outcomes, tests will be conducted in a controlled setting.
- **Defect Tracking:** Before the final report is turned in, any flaws discovered during testing will be noted, monitored, an.
- **Test Reporting:** Comprehensive test reports that include coverage metrics, pass/fail outcomes, and any problems encountered will be produced.

5.2 Metrics

The metrics used were:
Test Coverage: Unit and integration test coverage.
Bugs Detected/Resolved: Number of bugs detected and resolved during testing.
Performance Metrics: Response time, throughput under high load scenarios.
Mutation Testing Metrics: Number of mutations introduced and caught by the tests.
Graph-based Testing Metrics: Control flow and data flow coverage.
Logic-based Testing Metrics: Logical predicate and clause coverage.

5.3 Tools

The automated testing tools used: JUnit 4, TestNG, Pitclipse, and JMeter. Manual testing for CFG/DFG, ISP and Logic Based testing will be used.

5.4 Test Coverage

Test coverage for Unit Tests and Integration Tests was measured using functional coverage. The coverage goals for these tests are 100% and 80% respectively.

6. Conclusion & Reflection

The comprehensive testing of Apache Tika conducted successfully validated its core functionalities while uncovering critical areas for improvement. Through unit, integration, performance, mutation, input space partitioning (ISP), graph-based, and logic-based testing, we ensured the robustness, efficiency, and reliability of the toolkit

6.1 What Went Well

Achieved high unit test coverage (100%).

Effective integration between modules.

JUnit provided clear performance metrics.

Mutation testing improved test suite robustness.

6.2 Challenges & Improvements

Learning Curve: Initial difficulty with µJava and JMeter.

- **Improvement:** More time for tool familiarization.

Performance Bottlenecks: Large file handling.

- **Improvement:** Optimize buffer management.

Incomplete Graph-Based Testing:

- **Improvement:** Use automated CFG/DFG tools next time.

6.3 Tool Selection Justification

- **JUnit/TestNG: I**ndustry-standard for Java unit testing.
- **JMeter:** Best for load and integration testing.
- **Pitclipse:** Specialized for mutation testing in Java.
- **Manual ISP/Graph-Based:** Necessary for boundary and logic analysis.

This project not only reinforced the importance of multi-level testing but also highlighted how early defect detection saves development costs. While Apache Tika proved robust in most scenarios, performance tuning and edge-case handling remain key focus areas for future enhancements.

7. Final Approvals

Name	Student Number	Section	Signature*
Jason Zhao	500957196	6	JZ
Talha Hasan	501106651	6	TH
Venkat Guru Prasad	501114363	6	VG
Saad Bin Yunus	501017083	6	SBY
Patrick Urban	501034535	6	PU

The project requirements and the Test Plan Template are completely met by this Apache Tika test plan. It focuses on five distinct Apache Tika classes and contains all required sections, testing procedures, tools, and deliverables. This strategy complies with the course requirements and guarantees thorough testing of Apache Tika's essential features.

8. Contribution Matrix

Team Member	Assigned Testing Approach	Key Contributions	Tools Used	Deliverables
Jason Zhao	Logic-Based Testing	- Designed test cases for logical predicates in AutoDetectParser. - Identified false positives in MIME type detection. - Documented logic coverage gaps.	Manual Analysis, JUnit	Logic test cases, defect report, coverage analysis.
Talha Hasan	Integration Testing	- Configured JUnit for module interaction tests. - Tested parser processing time for large files - Analyzed throughput under load.	JUnit, TestNG	Integration test scripts, performance metrics, bug reports.
Saad Bin Yunus	Graph-Based Testing (CFG/DFG)	- Mapped control/data flow for chosen methods in all 5 classes. - Flagged unused variables in metadata processing. - Generated CFG/DFG diagrams. - Tested NC,EC,EPC,PPC,ADUPC, ADPC and AUPC	Manual Analysis	CFG/DFG diagrams, data flow analysis report.
Venkat Guru Prasad	Mutation Testing	- Implemented PITclipse mutations for core classes. - Evaluated test suite robustness (80% coverage). - Increased overall mutation coverage to 99% for core classes.	PITclipse, JUnit	Mutation test results, coverage gaps and improvements.
Patrick Urban	Input Space Partitioning (ISP)	- Defined boundary values for ISP tests - Uncovered crashes with malformed DOCX files. - Validated edge-case handling.	Manual Testing, JUnit	ISP test cases, boundary defect log, recommendations.

Collaborative Contributions

- **Environment Setup**:
  All members contributed to configuring Java 11, Maven, and Apache Tika.

- **Test Review**:
  Peer-reviewed test cases for correctness and completeness as per QA standards.

- **Report Drafting**:
  Co-authored and edited the final testing report with consolidated findings.

- **Tool Troubleshooting**:
  Collaboratively resolved issues encountered in JMeter and µJava configurations.

Effort Distribution

Activity	Percentage
Test Development	70%
Results Analysis	20%
Documentation	10%