Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu

SOFTWARE TESTING - APACHE COMMONS-JXPATH

1. INTRODUCTION

JXPath provides the API which is used to traverse the graphs in the JavaBeans, Maps, Servlets and DOM. The official language of XSLT is XPath. In case of XSLT if we want to access the various elemets of the XML documents we normally use XPath and sometimes this can be achieved by JXPath also. Along with this, it is also used to read and write various properties of JavaBeans and we can get and set the elements of the arrays, collections, maps, transparent containers and various context objects present in the Servlets. If we want to change the object models, JXPath uses the concepts of XPath. JXPath is also used to create new objects whenever it is necessary. To define an interpreter for the expression language XPath, a package org.apache.commons.jxpath is used.

Apache Commons JXPath is written in Java consisting of 200 classes and about 15 thousand lines of code.

Resources of Apache Commons JXPath used:
Github: https://github.com/apache/commons-jxpath.git
URL: https://commons.apache.org/proper/commons-jxpath/

2. POINTS ACHIEVED

TASK	POINT COVERAGE	PAGE NUMBERS
Building Project	10	2 - 4
Preliminary Test and Coverage	10	5
Functional Testing (FSM)	25	5 - 6
Static Analyzer (FindBugs)	15	7 - 10
Static Analyzer (PMD)	15	11 - 13
Reverse Eng (UML)	15	13 - 16
Reverse Eng (Call Graph Generator)	15	16 - 19
Mutation Testing (PIT)	15	19 – 21
Junit Test Case 20 LOC	5	21 - 23
TOTAL	**125**	

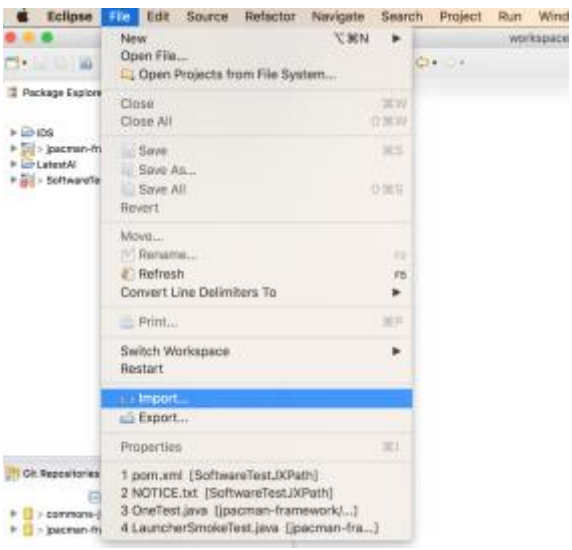
Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu

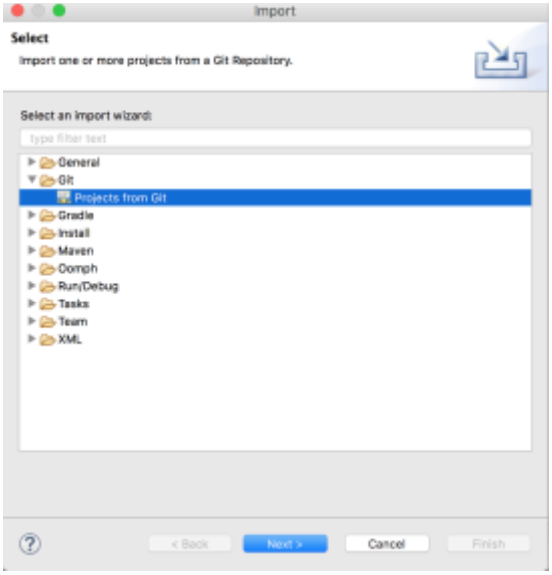
3. BUILDING THE PROJECT

For building the project, I have used Eclipse Java IDE. Apache Maven has also been installed. The source code of the project can be found in Github URL which I have specified.

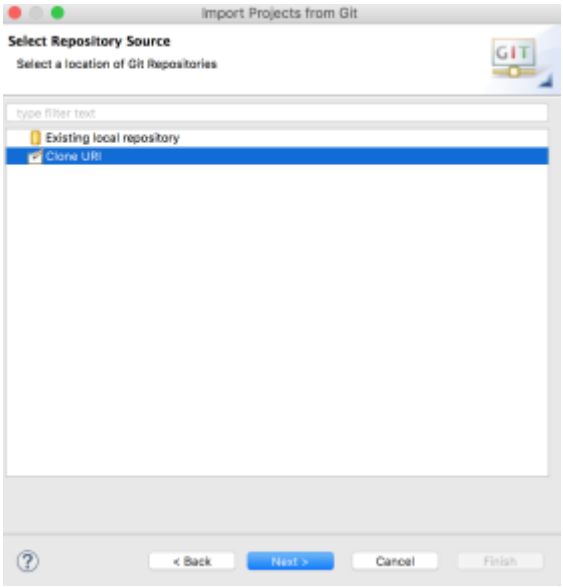
The installation steps are as follows:

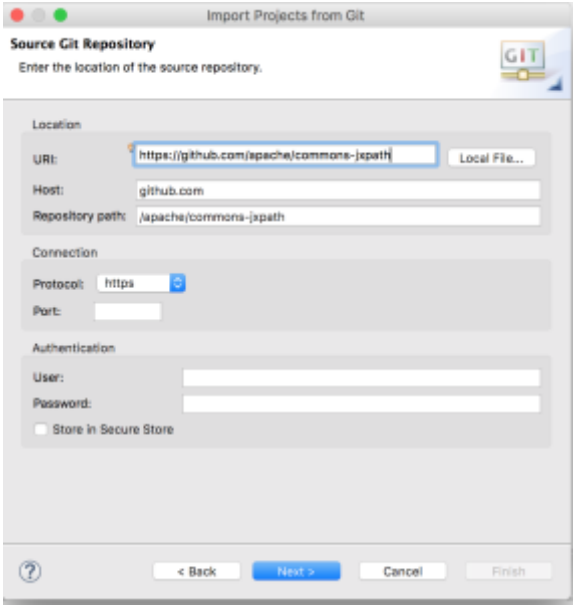
1. Open Eclipse and then go to import. In the import page select Project from Git.



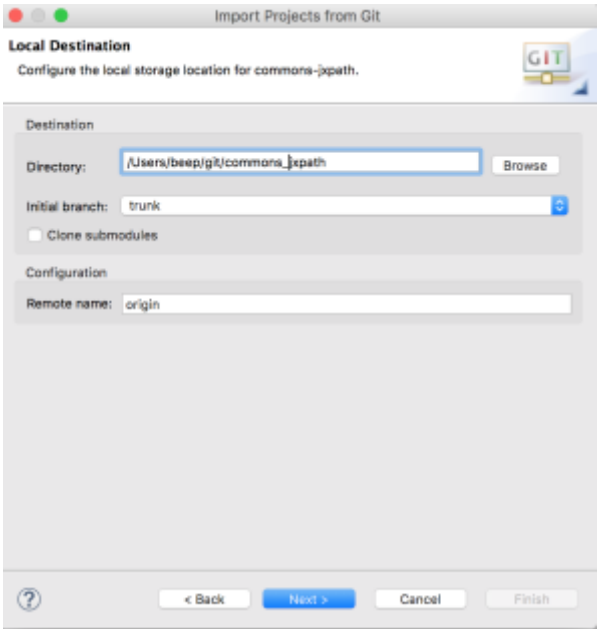


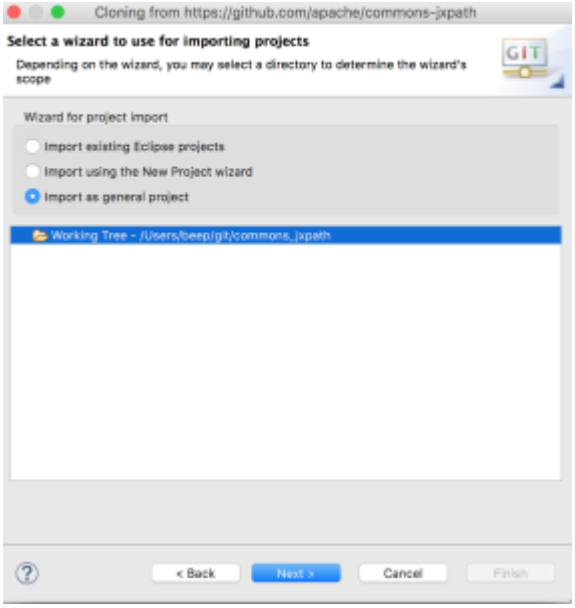
2. Then, clone the URL from Git. In the URL row select "https://github.com/apache/commons-jxpath.git". The host name and the repository path will be set to default which can be changed to our preference.



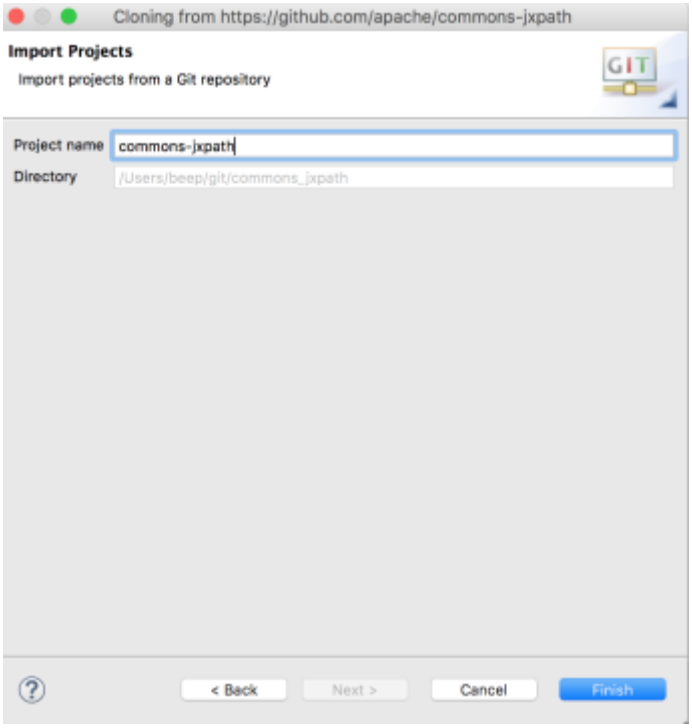


3. In the local destination, we need to give a path to the directory. It will be selected by default or we can change it to our specified path. In selecting a wizard used for importing the projects we need to select "Import as general project".

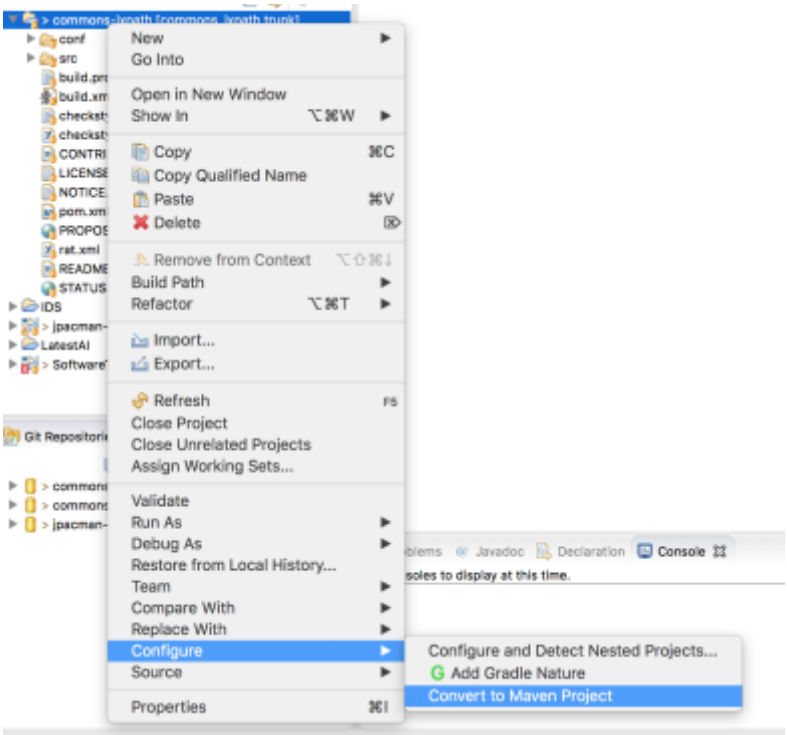




4. Next, a project name "commons-jxpath" is given in the Import Projects tab. The project name can be given as our choice as well. Next, we need to click on "Finish". Thus importing the project from Git to Eclipse is done successfully.



5. After successful importing we need to next convert it to maven. Hence, we need to right click on the main project and go to configure and then select Convert to Maven Project. Thus, the project is now configured to Maven. Next, click on the project and then build the entire project.

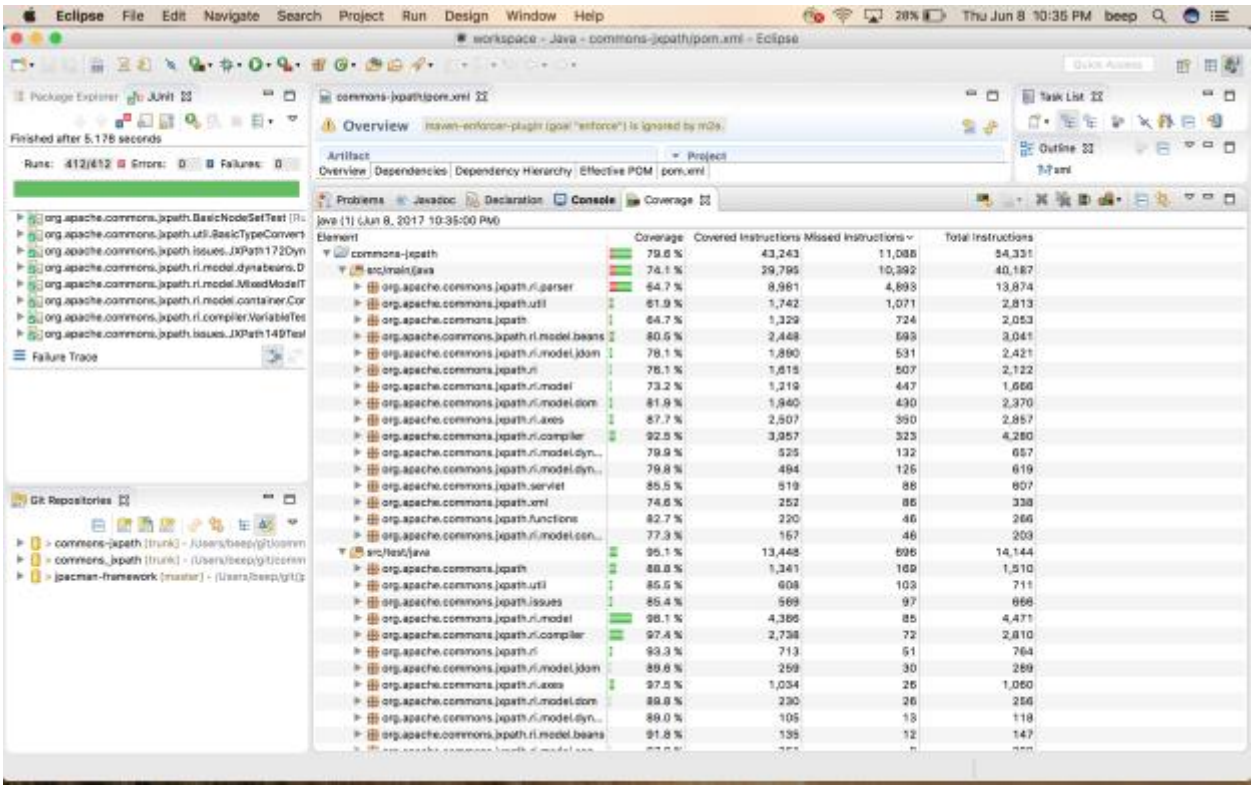


4. Preliminary testing and coverage

To perform preliminary testing and code coverage we need to install 2 tools namely Junit and EclEmma. EclEmma is tool used in eclipse for the Jave code coverage. It will launch directly within the workbench like the Junit test runs and analyses the code coverage in Eclipse. EclEmma can be downloaded in the Eclipse Marketplace.

From the below figure we can see that 79.6% of the code has been covered overall.

Element	Coverage	Covered Instructions	Missed Instructions
commons-jxpath	79.6%	43,243	11,088
src/main/java	74.1%	29,795	10,392
src/test/java	95.1%	13,448	696

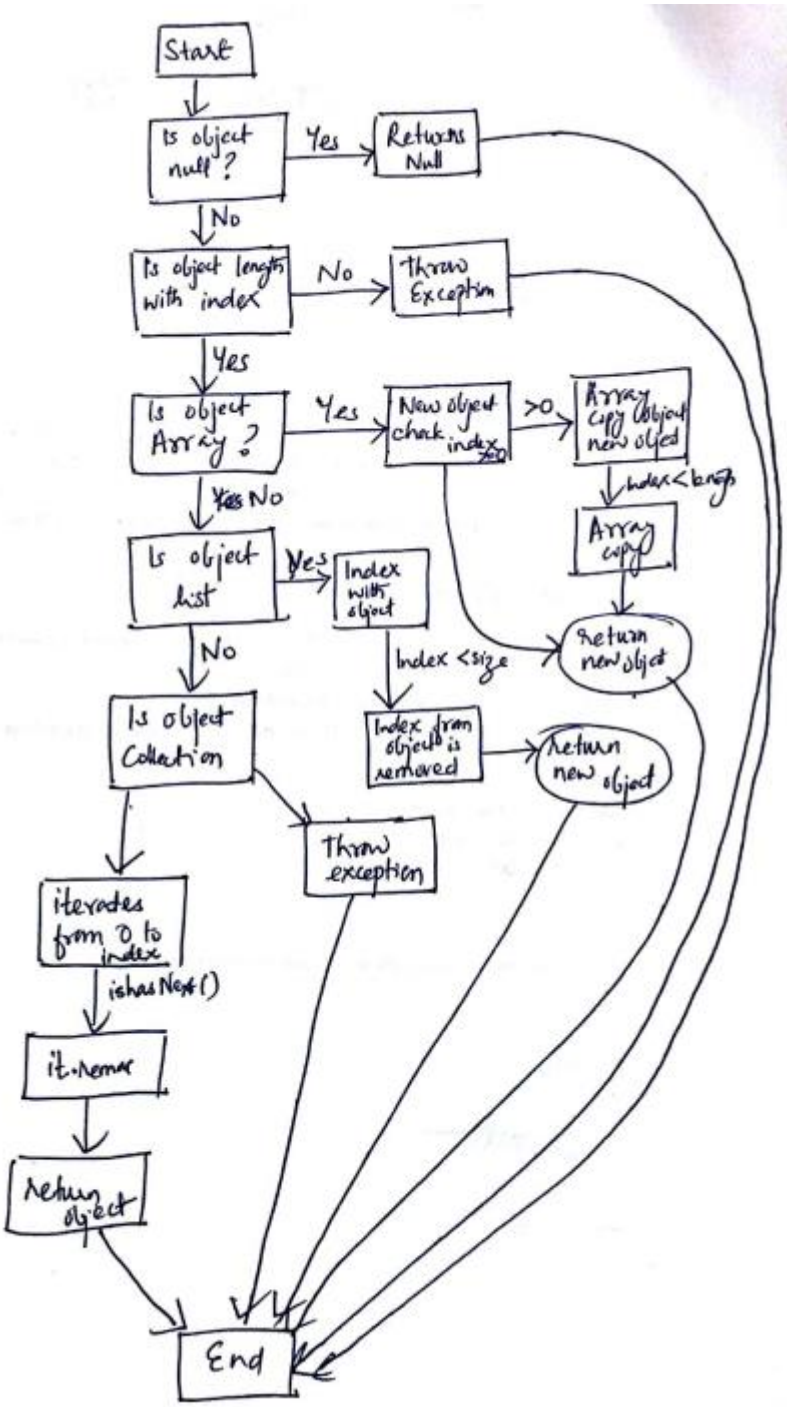


5. Functional testing (FSM)

The functional modeling can be done by using FSM aka **Finite State Machine**. The FSM seen in the below image are done in the different functions of the class.

In the below image we can see FSM model for the ValueUtils.java which is present in the org.apache.commons.jxpath.util package.

Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu



Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu

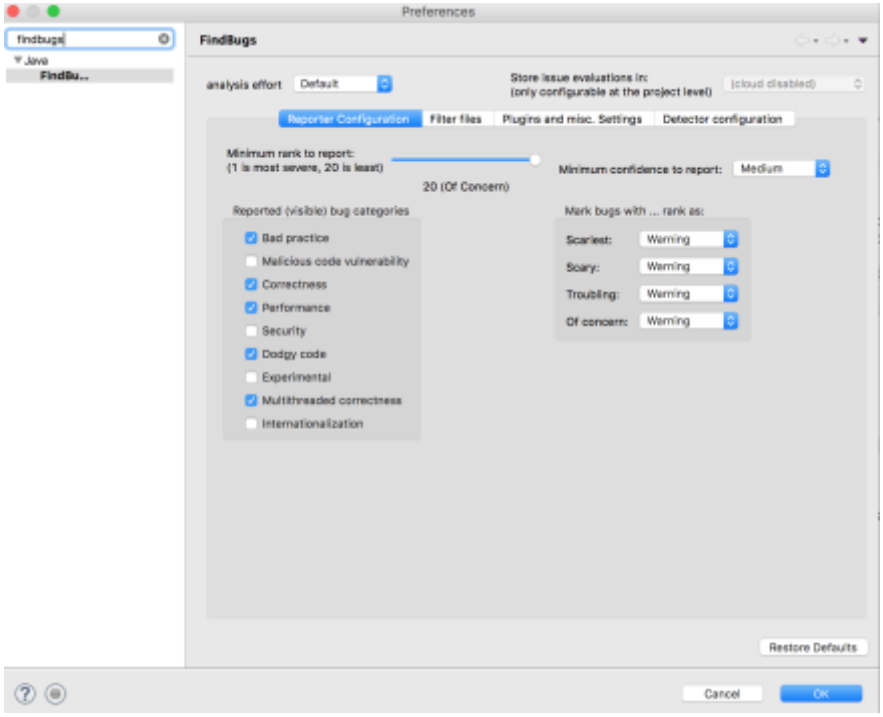
6. Structural Testing – Static Analyzer(FindBugs)

FindBugs is an open source tool which is used to identify the software bugs. It is beneficial for the developer in the early development phase. It analyses the Java Byte code and then identifies the software bugs so there may be a scenario that it might result in false positives. FindBugs is available as a plugin for Eclipse.

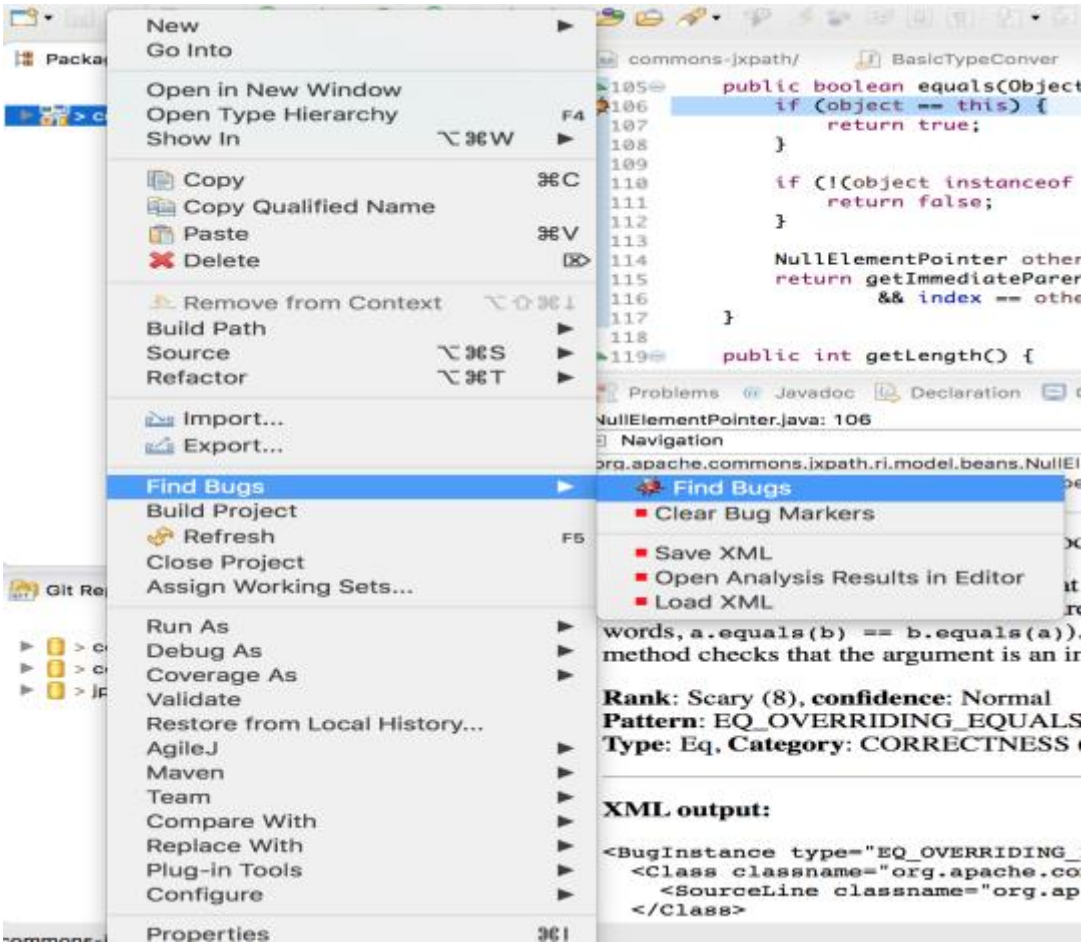
Installation and Setup

FindBugs can be installed in Eclipse by going to Help->Install New Software and then entering the URL(http://findbugs.cs.umd.edu/eclipse).

To increase the severity of the bugs we need to go to Eclipse -> Preferences and then search for FindBugs. In the reporter configuration tab, we need to change the "Minimum rank to report" to the maximum extent which is 20 as shown in the below figure. After making the necessary changes the setup is ready and we can run the FindBugs tool.



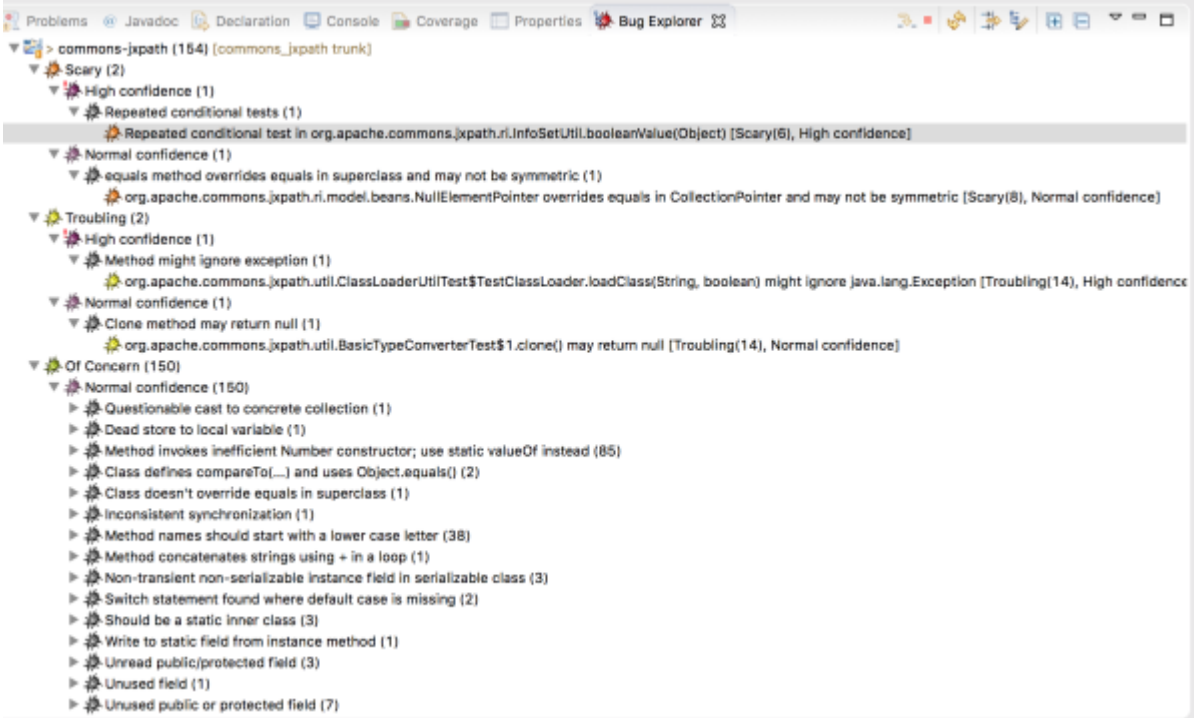
Next, we need to right click on the main project -> Find Bugs -> find bugs. This will in turn result in a bug report where we can see the statistics as shown in the below screenshots.



Below is the screenshot of the result after running FindBugs. From the below image, we can see the following report for Apache Commons JXPath:

- Total number of Bugs in Commons JXPath – 154
- Scary – 2
    - High confidence – 1
    - Normal confidence - 1
- Troubling – 2
    - High confidence – 1
    - Normal confidence - 1
- Of Concern – 150
    - Normal confidence - 150

Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu



Justifiable scenario

In the picture attached below we see the Bug Info for the repeated conditional test. In the navigation tab, it says that the conditional test has been repeated multiple times and hence there is a bug due to that. It also provides the rank of the bug, confidence, pattern, type and category of the bug statistics. It also provides the output in the form of XML.

It has determined the bug through Find Bugs because the conditional test is performed twice here. The second occurrence of value not equal to 0 can be avoided per the Bug Info.

Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu

```
140            return value != 0 && value != negZero && !Double.isNaN(value);
141        }
142        if (object instanceof Boolean) {
143            return ((Boolean) object).booleanValue();
```

Problems | Javadoc | Declaration | Console | Coverage | Properties | Bug Explorer | Bug Info

InfoSetUtil.java: 140

Navigation

Repeated conditional test in org.apache.commons.jxpath.ri.InfoSetUtil.booleanValue(Object)

Bug: Repeated conditional test in org.apache.commons.jxpath.ri.InfoSetUtil.booleanValue(Object)

The code contains a conditional test is performed twice, one right after the other (e.g., `x == 0 || x == 0`). Perhaps the second occurrence is intended to be something else (e.g., `x == 0 || y == 0`).

Rank: Scary (6), **confidence**: High
Pattern: RpC_REPEATED_CONDITIONAL_TEST
Type: RpC, **Category**: CORRECTNESS (Correctness)

XML output:

```
<BugInstance type="RpC_REPEATED_CONDITIONAL_TEST" priority="1" rank="6" abbrev="RpC" category="CORRECTNESS" first="1">
  <Class classname="org.apache.commons.jxpath.ri.InfoSetUtil">
    <SourceLine classname="org.apache.commons.jxpath.ri.InfoSetUtil" sourcefile="InfoSetUtil.java" sourcepath="org/apache
  </Class>
  <Method classname="org.apache.commons.jxpath.ri.InfoSetUtil" name="booleanValue" signature="(Ljava/lang/Object;)Z" isSt
    <SourceLine classname="org.apache.commons.jxpath.ri.InfoSetUtil" start="137" end="161" startBytecode="0" endBytecode=
  </Method>
  <SourceLine classname="org.apache.commons.jxpath.ri.InfoSetUtil" start="140" end="140" startBytecode="17" endBytecode="
  <SourceLine classname="org.apache.commons.jxpath.ri.InfoSetUtil" start="140" end="140" startBytecode="23" endBytecode="
</BugInstance>
```

False positive warning

In the below code snippet, we can see that the function Booleanvalue will return value != 0 && value != negZero. The error is caused by FindBugs because conditional test is being performed two times. Once when the value != 0 and another when value != negZero. Here negZero is assigned the value of -0. In the normal mathematical computation both 0 and -0 are treated as equal but in this scenario, those two operations are treated as different. Hence this is a false positive warning caused by FindBugs.

```
136    public static boolean booleanValue(Object object) {
137        if (object instanceof Number) {
138            double value = ((Number) object).doubleValue();
139            final int negZero = -0;
140            return value != 0 && value != negZero && !Double.isNaN(value);
141        }
142        if (object instanceof Boolean) {
143            return ((Boolean) object).booleanValue();
144        }
145        if (object instanceof EvalContext) {
146            EvalContext ctx = (EvalContext) object;
147            Pointer ptr = ctx.getSingleNodePointer();
148            return ptr == null ? false : booleanValue(ptr);
149        }
150        if (object instanceof String) {
```

Problems | Javadoc | Declaration | Console | Coverage | Properties | Bug Explorer | Bug Info

- commons-jxpath (154) [commons_jxpath trunk]
  - Scary (2)
    - High confidence (1)
      - Repeated conditional tests (1)
        - Repeated conditional test in org.apache.commons.jxpath.ri.InfoSetUtil.booleanValue(Object) [Scary(6), High confidence]

Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu

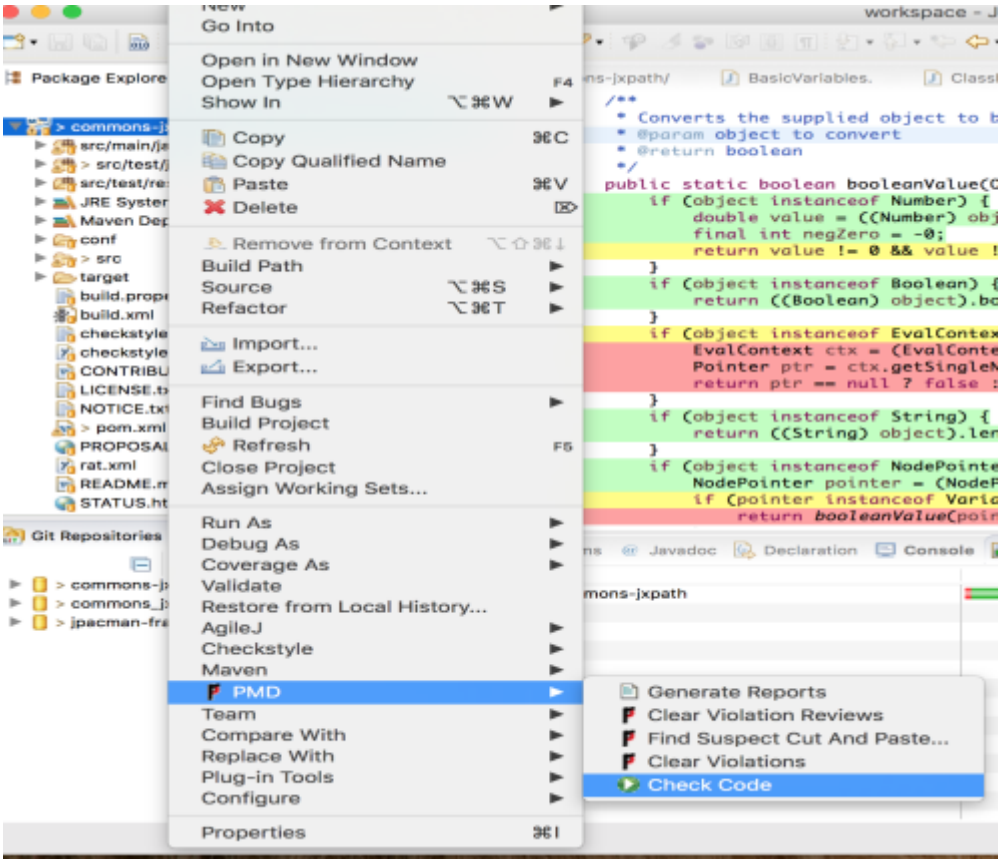
7. Structural Testing – Static Analyzer(PMD)

PMD aka Programming Mistake Detector is a free source tool which is used to analyze the software bug and helps in improving the quality of the code. It is used to determine the bugs such as if there is unnecessary creation of the objects, empty cases in if/else, try, catch, finally, while and switch statements in the Java code.

Installation and Setup

PMD can be installed in Eclipse by going to Help->Install New Software and then entering the URL([https://dl.bintray.com/pmd/pmd-eclipse-plugin/updates/)](https://dl.bintray.com/pmd/pmd-eclipse-plugin/updates/) and giving the name as PMD for Eclipse Update Site.

To run the PMD, we need to right click on the main project -> PMD -> check code as shown in the screenshot below.



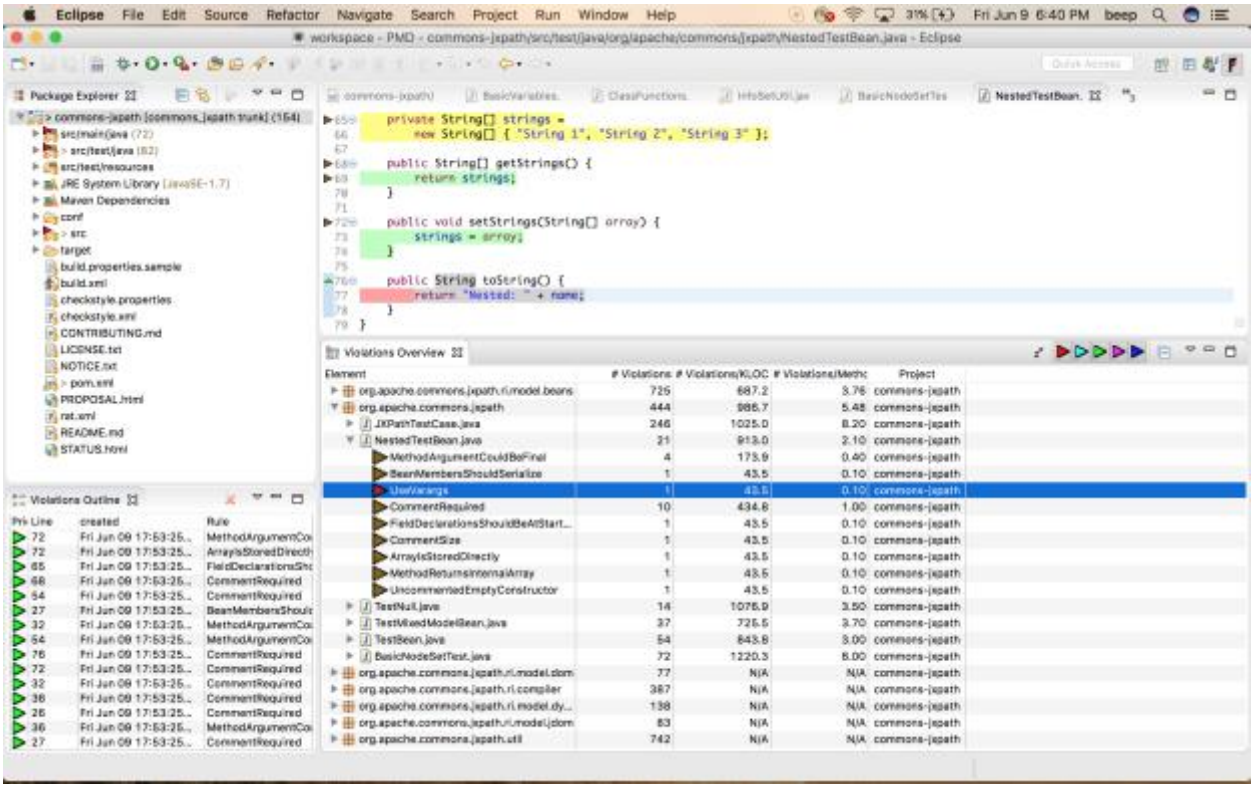
After running PMD, we can see the package level overview in the Violations overview in the below figure. It consists of the number of violations, violations per kilo lines of code, violations per method and the project description for each package. These violations which are displayed will be helpful while debugging since these are warnings and not bugs.

Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu

Violations Overview

Element	# Violations	# Violations/KLOC	# Violations/Methc	Project
org.apache.commons.jxpath.ri.model.beans	725	N/A	N/A	commons-jxpath
org.apache.commons.jxpath	444	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.dom	77	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.compiler	387	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.dy...	138	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.jdom	83	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.util	742	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.axes	721	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.parser	3202	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model	896	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.dom	571	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.servlet	212	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.functions	69	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.dy...	40	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.util	179	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.issues	259	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model	344	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.xml	80	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.dy...	136	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.servlet	86	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.co...	69	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri	579	N/A	N/A	commons-jxpath
org.apache.commons.jxpath	570	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.beans	52	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.dy...	121	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri	168	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.axes	177	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.co...	70	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.compiler	1240	N/A	N/A	commons-jxpath
org.apache.commons.jxpath.ri.model.jdom	586	N/A	N/A	commons-jxpath

In the below screenshot, we see that display of violations per each class.



Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu

Pros/Cons of Find Bugs and PMD

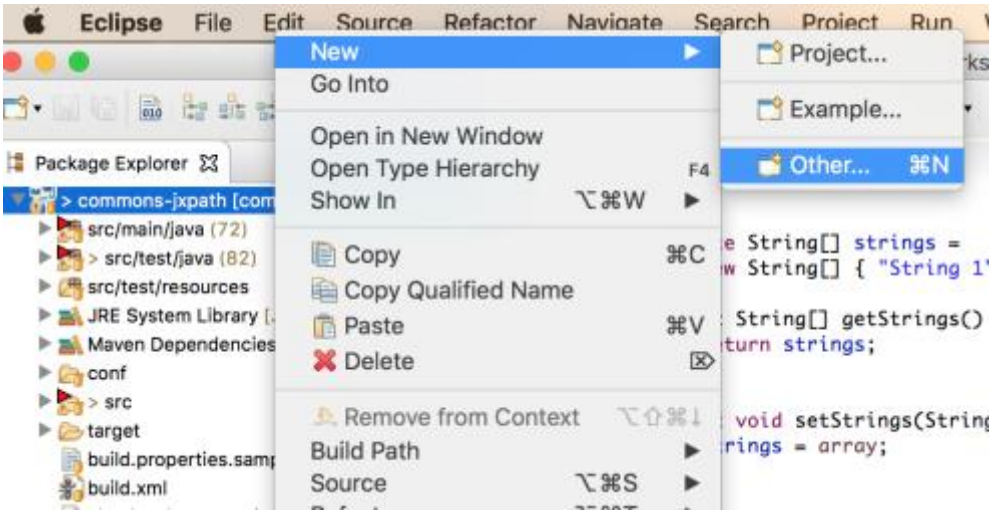
Find Bugs	PMD
For the analysis, it operates on byte code.	For the analysis, it operates on source code.
Most of the time it finds real bugs.	Rarely it finds real bugs.
Find Bugs requires code to be compiled.	PMD does not require code to be compiled.
False detection rate is low.	Fast detection rate is not low.
It is fast because it uses byte code.	It is slow because it works on source code.
Custom rules addition in Find Bugs is difficult.	Custom rules addition is easy and simple in case of PMD.
In Find Bugs, the equals() method fails on subtypes, clone method may return null, reference comparison of Boolean values, impossible cast.	In PMD, there are violation of naming conventions, lack of curly braces, misplaced null check, long parameter list, unnecessary constructor, missing break in switch statements.

8. Reverse Engineering Tools (UML diagram generator)

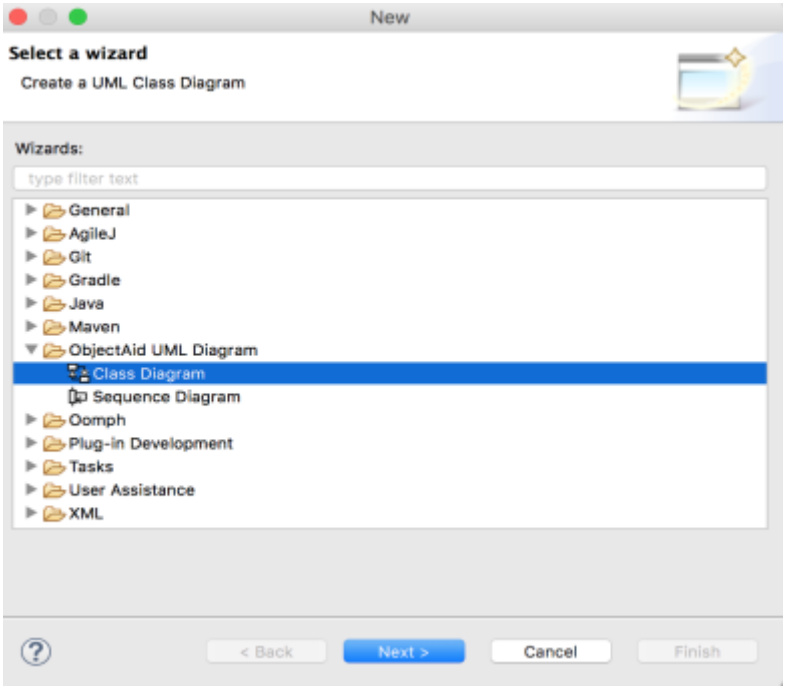
To generate UML diagram generator, **Object AID** tool is used.

Installation and Setup

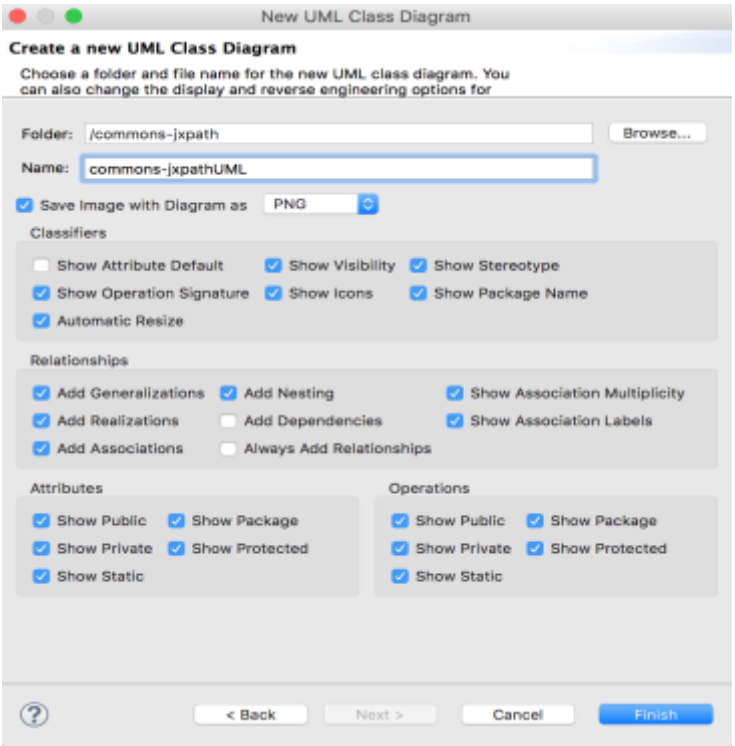
Go to Eclipse MarketPlace which is there in the Help tab in the Eclipse. Then enter the Object AID and the necessary tool needs to be installed. After installing Object AID successfully, right click on the entire package -> New -> Other as shown below.



Then, click on the Object AID UML diagram and select the Class Diagram tab and press next as shown in the below screenshot.

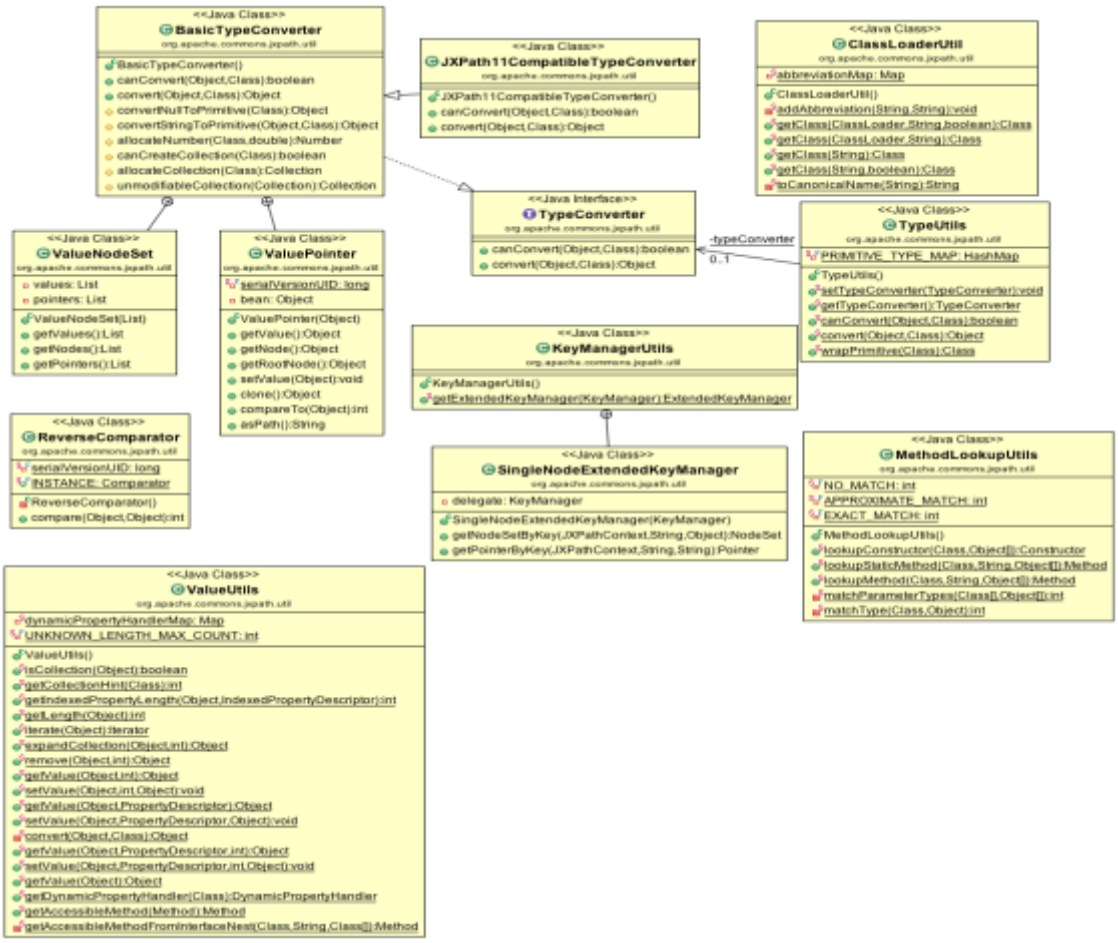


After clicking next, create new UML class diagram window will open and we need to enter a name we need to give to the UML Class Diagram. In the below figure, - has been given as the name. Then enter Finish.

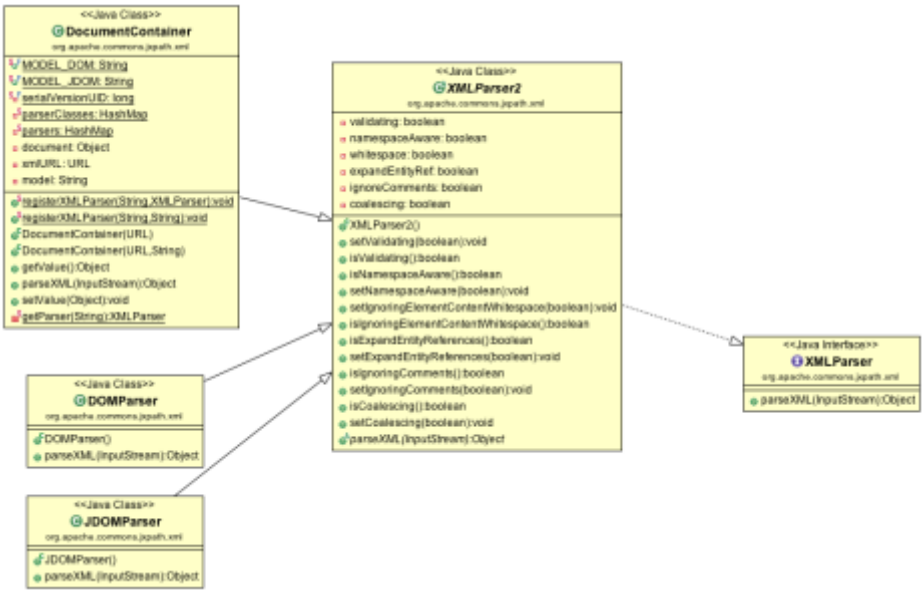


Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu

Below is the UML class diagram which is generated for org.apache.commons.jxpath.util



Below is the UML class diagram which is generated for org.apache.commons.jxpath.xml



Use of UML diagram for testing

UML class diagram will provide the insight about a new member of a project to understand the dependencies about different classes in a package. This also helps in understanding the structure of the code and is easier to understand. Since the understanding of the code is easier it helps in providing a better test cases and hence improves the efficiently of the code.

The files which are generated in UML diagram are as follows:

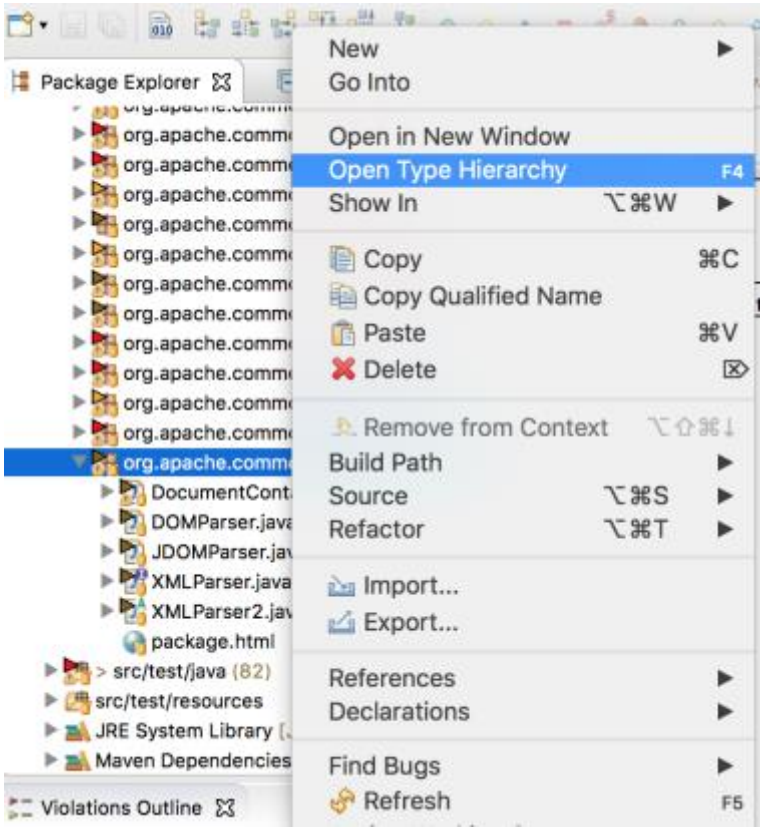
- org.apache.commons.jxpath.util – utilUML.ucls
- org.apache.commons.jxpath.xml – uml2.ucls
- org.apache.commons.jxpath.ri.model – commons-jxpathUML.ucls

9. Reverse Engineering Tool (Call Graph Generator)

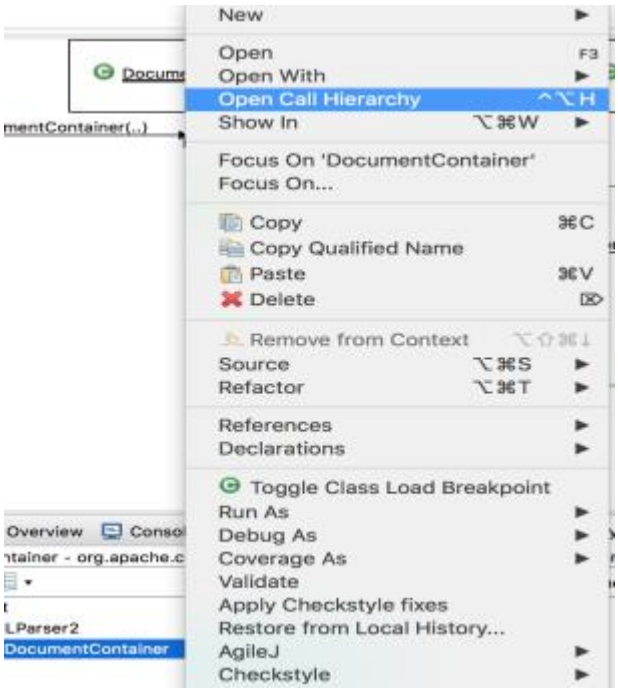
To generate Call Graph Generator, **CallGraph Viewer** is used.

Installation and Setup

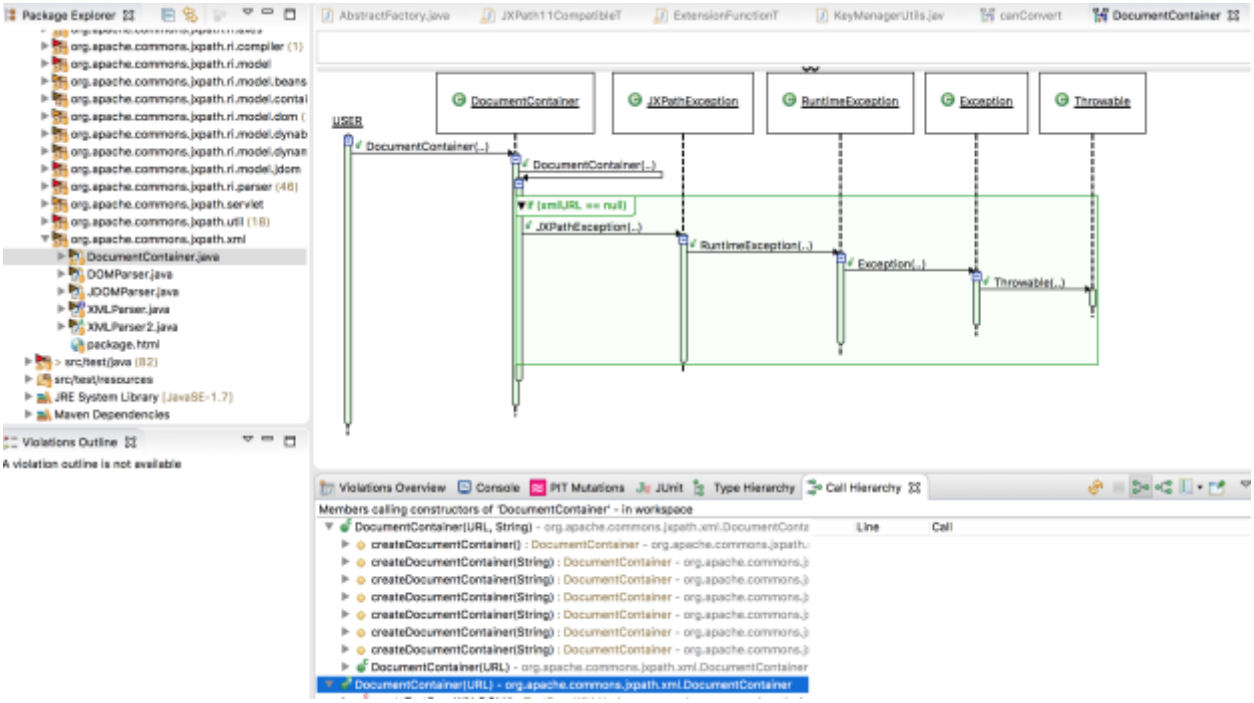
Go to Eclipse MarketPlace which is there in the Help tab in the Eclipse. Then enter the CallGraph Viewer and the necessary tool needs to be installed. After installing CallGraph Viewer successfully, right click on one of the packages and click "Open New Type Hierarchy" as shown in the figure.



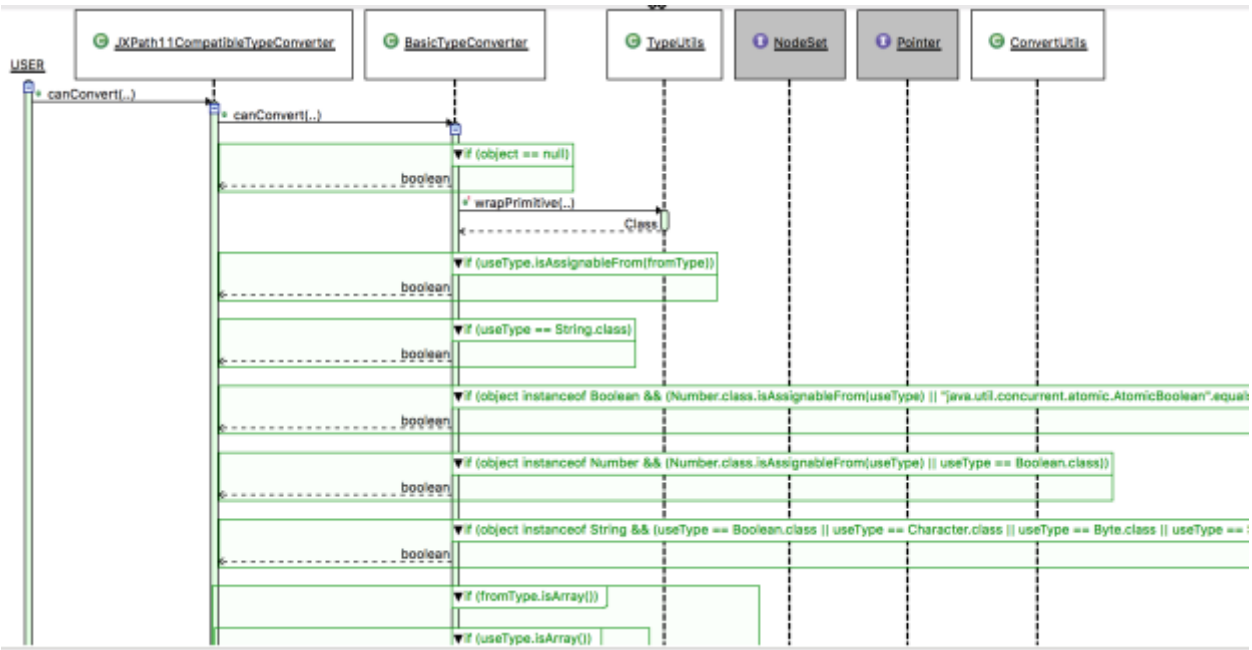
Next, you enter the type hierarchy tab and we need to right click on one of the java files and click on “Call Hierarchy” as shown in the figure.



After this, we need to enter to one of the methods in the java files, right click on the function name and click “Sequence Diagram” to generate. Below is the sequence diagram of DocumentContainer.java.



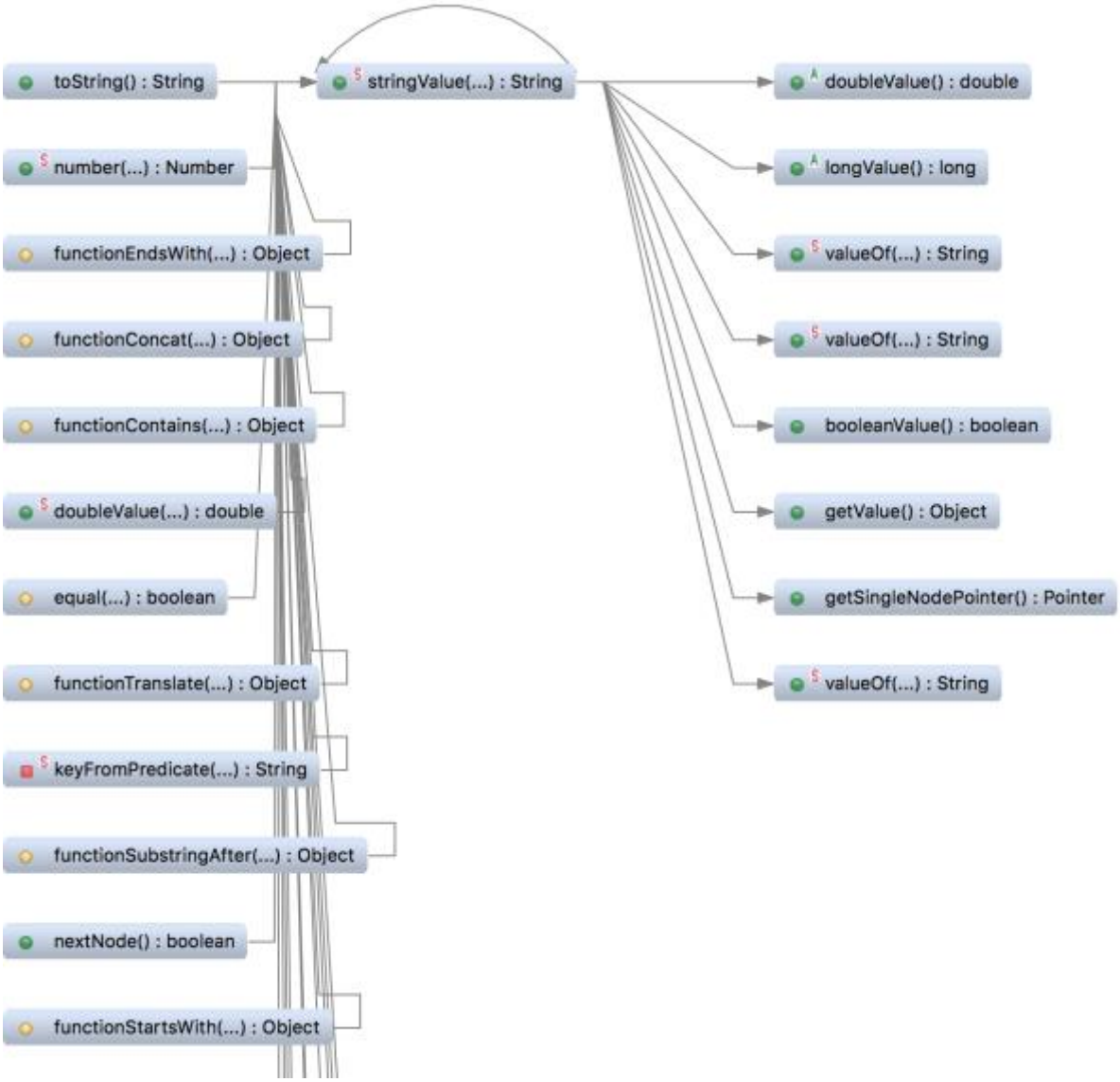
In the below screenshot we can see the sequence diagram of JXPath Compatible Path Generator.



Use of Call Graph Viewer for testing

Call Graph viewer provides sequence diagrams and call path for the users. It provides graphical view of all the other methods which are related to each other and hence it plays an important role for a new member in the team to understand the call hierarchies and sequence of flow between the methods in a class. Since all the call hierarchies will be understood clearly, it is easier to write better test cases and hence helps in the efficiency of the code.

To generate the Call Graph, we need to right click on any method name and then click Call graph -> Add Callers and then click Add Callees as shown in the below figure.



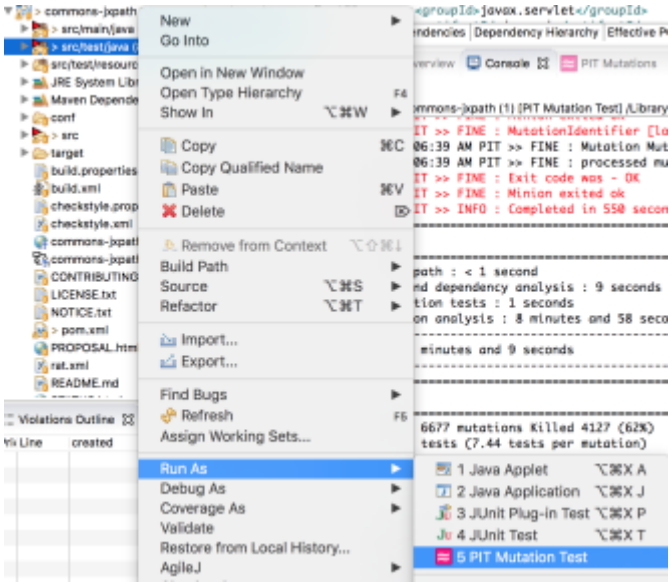
10. Program Analysis – Mutation Testing(PIT)

The mutation tool in Java for Eclipse IDE is called as **PIT**. It is used to measure the unit test coverage. The mutation score is 62% and line coverage is 77%

Installation and Setup

PITclipse can be installed from the Eclipse Marketplace. After the installation of the PITclipse, we need to go to src/test/java and then right click and then go to Run As -> PIT Mutation Test as shown below in the figure.

Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu



Below is the PIT Summary status after successful compilation.

Violations Overview | Console | PIT Mutations | JUnit | PIT Summary | Type Hierarchy | Call Hierarchy

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
142	77%	7727/10019	62%	4127/6677

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.apache.commons.jxpath	14	68%	405/598	60%	151/250
org.apache.commons.jxpath.functions	2	82%	53/65	93%	38/41
org.apache.commons.jxpath.ri	8	77%	448/582	78%	243/312
org.apache.commons.jxpath.ri.axes	14	90%	806/895	76%	418/551
org.apache.commons.jxpath.ri.compiler	33	92%	1004/1090	83%	650/785
org.apache.commons.jxpath.ri.model	3	75%	307/411	76%	217/287
org.apache.commons.jxpath.ri.model.beans	16	82%	651/797	73%	421/576
org.apache.commons.jxpath.ri.model.container	2	78%	35/45	47%	21/45
org.apache.commons.jxpath.ri.model.dom	7	81%	490/607	74%	303/411
org.apache.commons.jxpath.ri.model.dynabeans	4	82%	120/147	71%	67/95
org.apache.commons.jxpath.ri.model.dynamic	4	83%	141/169	68%	72/106
org.apache.commons.jxpath.ri.model.jdom	7	77%	486/631	71%	302/425
org.apache.commons.jxpath.ri.parser	6	69%	1984/2877	39%	848/2184
org.apache.commons.jxpath.servlet	10	88%	184/210	86%	61/71
org.apache.commons.jxpath.util	8	67%	526/787	60%	302/501
org.apache.commons.jxpath.xml	4	81%	87/108	35%	13/37

In the next image, we can see the PIT mutations status.

Violations Overview | Console | PIT Mutations | JUnit | PIT Summary | Type Hierarchy | Call Hierarchy

- SURVIVED (971)
  - commons-jxpath (971)
    - org.apache.commons.jxpath (27)
      - org.apache.commons.jxpath.BasicNodeSet (4)
        - 41: negated conditional
        - 42: removed call to org/apache/commons/jxpath/BasicNodeSet::clearCacheLists
        - 51: negated conditional
        - 52: removed call to org/apache/commons/jxpath/BasicNodeSet::clearCacheLists
      - org.apache.commons.jxpath.FunctionLibrary (1)
        - 90: changed conditional boundary
      - org.apache.commons.jxpath.JXPathContext (9)
        - 572: negated conditional
        - 572: negated conditional
        - 657: replaced return of integer sized value with (x == 0 ? 1 : 0)
        - 672: negated conditional
        - 714: mutated return of Object value for org/apache/commons/jxpath/JXPathContext::selectNodes to (if (x != null) null else throw new RuntimeException)
        - 842: negated conditional
        - 842: negated conditional
        - 879: negated conditional
        - 879: negated conditional
      - org.apache.commons.jxpath.JXPathContextFactory (5)
        - 164: negated conditional
        - 187: negated conditional
        - 224: negated conditional
        - 231: negated conditional
        - 264: mutated return of Object value for org/apache/commons/jxpath/JXPathContextFactory::findFactory to (if (x != null) null else throw new RuntimeException)
      - org.apache.commons.jxpath.JXPathException (4)
        - 88: negated conditional
        - 92: negated conditional
        - 96: negated conditional
        - 97: mutated return of Object value for org/apache/commons/jxpath/JXPathException::getMessage to (if (x != null) null else throw new RuntimeException)
      - org.apache.commons.jxpath.JXPathIntrospector (2)
        - 88: negated conditional

Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

Violations Overview | Console | PIT Mutations | JUnit | PIT Summary | Type Hierarch

```
<terminated> commons-jxpath (1) [PIT Mutation Test] /Library/Java/JavaVirtualMachines/jdk1.8.0_111.jdk/Contents/Home/bin/ja
1:06:39 AM PIT >> FINE : Exit code was - OK
1:06:39 AM PIT >> FINE : Minion exited ok
1:06:39 AM PIT >> FINE : MutationIdentifier [location=Location [clazz=org.apache.commons.jxpath.r
stderr  : 1:06:39 AM PIT >> FINE : Mutation MutationIdentifier [location=Location [clazz=org.apac
stderr  : 1:06:39 AM PIT >> FINE : processed mutation in 412 ms.
1:06:39 AM PIT >> FINE : Exit code was - OK
1:06:39 AM PIT >> FINE : Minion exited ok
1:06:39 AM PIT >> INFO : Completed in 550 seconds
================================================================================
- Timings
================================================================================
> scan classpath : < 1 second
> coverage and dependency analysis : 9 seconds
> build mutation tests : 1 seconds
> run mutation analysis : 8 minutes and 58 seconds
--------------------------------------------------------------------------------
> Total  : 9 minutes and 9 seconds
--------------------------------------------------------------------------------
================================================================================
- Statistics
================================================================================
>> Generated 6677 mutations Killed 4127 (62%)
>> Ran 49676 tests (7.44 tests per mutation)
================================================================================
- Mutators
================================================================================
> org.pitest.mutationtest.engine.gregor.mutators.ConditionalsBoundaryMutator
>> Generated 319 Killed 154 (48%)
> KILLED 154 SURVIVED 70 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 95
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.IncrementsMutator
>> Generated 166 Killed 109 (66%)
> KILLED 103 SURVIVED 7 TIMED_OUT 6 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
```

Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu

11. Structural Testing - Junit Test Cases to improve Code Coverage

Test case 1:

Package: org.apache.commons.jxpath.ri.parser
Changed File: ParserException.java
Test Case Package: mytests
Test Case File Name: ParserTest.java
Lines improved: 28

Before writing the Test Case (snippet of the code and the coverage):

ParseException. | ParserTest.java | XPathParserToke | XPathParserCons

```
146     */
147   public String add_escapes(String str) {
148       StringBuffer retval = new StringBuffer();
149       char ch;
150       for (int i = 0; i < str.length(); i++) {
151         switch (str.charAt(i))
152         {
153            case 0 :
154               continue;
155            case '\b':
156               retval.append("\\b");
157               continue;
158            case '\t':
159               retval.append("\\t");
160               continue;
161            case '\n':
162               retval.append("\\n");
163               continue;
164            case '\f':
165               retval.append("\\f");
166               continue;
167            case '\r':
168               retval.append("\\r");
169               continue;
170            case '\"':
171               retval.append("\\\"");
172               continue;
173            case '\'':
174               retval.append("\\\'");
175               continue;
176            case '\\':
177               retval.append("\\\\");
178               continue;
179            default:
```

XPathParser.java	77.6 %	4,561	1,319	5,880
SimpleCharStream.java	32.3 %	315	661	976
ParseException.java	0.0 %	0	375	375
XPathParserConstants.java	0.0 %	0	364	364
TokenMgrError.java	0.0 %	0	183	183
Token.java	75.0 %	9	3	12
org.apache.commons.jxpath.util	61.9 %	1,742	1,071	2,813

Deepthi Devaiah Devanira (65848248)
Email id: ddevanir@uci.edu

After writing the Test Case (snippet of the code and the coverage):

ParseException. | ParserTest.java | XPathParserToke | XPathParserCons | »12

```
146     */
147   public String add_escapes(String str) {
148       StringBuffer retval = new StringBuffer();
149       char ch;
150       for (int i = 0; i < str.length(); i++) {
151         switch (str.charAt(i))
152         {
153            case 0 :
154               continue;
155            case '\b':
156               retval.append("\\b");
157               continue;
158            case '\t':
159               retval.append("\\t");
160               continue;
161            case '\n':
162               retval.append("\\n");
163               continue;
164            case '\f':
165               retval.append("\\f");
166               continue;
167            case '\r':
168               retval.append("\\r");
169               continue;
170            case '\"':
171               retval.append("\\\"");
172               continue;
173            case '\'':
174               retval.append("\\\'");
175               continue;
176            case '\\':
177               retval.append("\\\\");
178               continue;
179            default:
```

XPathParserTokenManager.java	67.3 %	4,096	1,988	6,084
SimpleCharStream.java	32.3 %	315	661	976
ParseException.java	24.0 %	90	285	375
Token.java	75.0 %	9	3	12
TokenMgrError.java	0.0 %	0	183	183