!!! NOTE TO GRADER !!! This report is written in Github flavored markdown, then converted to pdf. For a better readability, please use markdown readers or view it on github. Here is the link: https://github.com/snklee/mutation-analysis/blob/master/README.md

Mutation analysis of Jsoup project

Jsoup as target of analysis

about the project

Jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods. It's authored by Jonathan Hedley, a software development manager for Amazon Seattle, and distributed under the open source MIT license. Here's the jsoup's website and github page.

Source code and build environment

Jsoup had 6120 lines of Java code for testing purposes and 18709 lines of Java code for functionality at the time of this analysis. The test suit uses JUnit with ant integration and analysis is done in Ubuntu 14.04 system.

Running Major with the jsoup

Tests and mutations

The project had 469 tests in total. With `all.mml`, 6684 mutants are generated. It took roughlt 3 hours for my laptop to finish the mutation analysis for the project.

Reproduce the analysis results

I have extracted the `build.xml` and `maven-build.xml` with command `mvn ant:ant`, and modified `build.xml` configuration to use Major. You can reproduce the full mutation analysis result by executed shell script `run.sh`. Alternatively, you may execute `smallSampleRun.sh`, to generate small amount of mutants, with smaller.mml file, to get a quick result.

About files generated

Here are a list of files generated by the run: * **mutants.log** contains one line for each generated mutant. Each line contains the type of mutation, the method and line of the mutation, and the actual syntactic mutation applied. * **killed.csv** contains a line for each generated mutant. Each line identifies the mutant ID from mutants.log and the status of the mutant at the end of analysis. FAIL, TIME, and EXC mean that the mutant was killed by an assertion failure, test timeout, or an exception. LIVE means that the mutant was still live at the end of analysis. * **results.csv** contains the mutation analysis for each individual test in the test suite. * **testMap.csv** contains mapping of test number shown in result.csv and test name. * **summary.csv** contains the complete mutation

analysis results. * **stdout.log** contains logs written to stdout in the during the run. * **target** complied *.class files for the project

Result and analysis

Result in numbers

- The number of mutants generated = **6684**
- The number of mutants covered by the test suite = **5398** (80.76% of generate)
- The number of mutants killed by the test suite = **3588**
- The number of live mutants = **1810**
- The overall mutation score(kill/generated) / adequacy of the test suite(killed/covered) = **53.68% (66.47%)**

The result shows that `killed + live = covered`. Covered means a mutant occured in a line that get executed by at least one of the unit tests. If the unit tests do not even execute that line, then the test suite definitely cannot kill that mutant. On the other hand, if a mutant is covered, the test suite might or might not kill the mutant, depending on the strangth of the test suite.

The mutation analysis showed the quality of the test suite. On the upside, I was able to integrate the tool (Major) with a project exited for sometime in a short time. On the downside, each run of the analysis tooks far more time than unit testing, making it not suitable for continious integration.

For someone who has never used Maven, ant nor JUnit, let alone Major, the process was fairly smooth. A major obstacle was to modify the `build.xml` file. I overcame it by learning from the sample file and online documentations.

Test suite performance

The coverage score (80.76%) shows almost 20% of the code are not covered by any test case. This means whatever bug introduced to that part will not be detected by the test suite. This means tests cases need to be added to the test suite to cover the one fifth of the code base. The kill/covered(66.47%) score shows how good is the test suite on detecting a bug introduced in the covered code base. The scores indicates that test cases needs to be improved to provide better capability to detect bug.

Analysis of mutants

In this section, I will look into 25 selected mutants within a method, to give a detailed description of what happend.

Here is the target method selected:

```
225    String consumeHexSequence() {
226        int start = pos;
227        while (pos < length) {
```

```
228             char c = input[pos];
229             if ((c >= '0' && c <= '9') || (c >= 'A' && c <= 'F') || (c >= 'a' &&
c <= 'f'))
230                 pos++;
231             else
232                 break;
233         }
234         return cacheString(start, pos - start);
235     }
```

Original line from mutants.log:

Mutant Number	Mutant
1600	<(int,int):!=(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:227:pos < length |==> pos != length
1601	<(int,int):<=(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:227:pos < length |==> pos <= length
1602	<(int,int):FALSE(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:227:pos < length |==> false
1603	>=(int,int):==(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= '0' |==> c == '0'
1604	>=(int,int):>(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= '0' |==> c > '0'
1605	>=(int,int):TRUE(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= '0' |==> true
1606	<=(int,int):<(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c <= '9' |==> c < '9'
1607	<=(int,int):==(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c <= '9' |==> c == '9'
1608	<=(int,int):TRUE(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c <= '9' |==> true
1609	&&(boolean,boolean):==(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= '0' && c <= '9' |==> c >= '0' == c <= '9'
1610	&&(boolean,boolean):FALSE(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= '0' && c <= '9' |==> false
1611	&&(boolean,boolean):LHS(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= '0' && c <= '9' |==> c >= '0'
1619	&&(boolean,boolean):==(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= 'A' && c <= 'F' |==> c >= 'A' == c <= 'F'
1620	&&(boolean,boolean):FALSE(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= 'A' && c <= 'F' |==> false
1621	&&(boolean,boolean):LHS(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= 'A' && c <= 'F' |==> c >= 'A'
1622	&&(boolean,boolean):RHS(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= 'A' && c <= 'F' |==> c <= 'F'
1623	||(boolean,boolean):!=(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:(c >= '0' && c <= '9') || (c >= 'A' && c <= 'F') |==> (c >= '0' && c <= '9') != (c >= 'A' && c <= 'F')
1624	||(boolean,boolean):LHS(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:(c >= '0' && c <= '9') || (c >= 'A' && c <= 'F') |==> (c >= '0' && c <= '9')
1625	||(boolean,boolean):RHS(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:(c >= '0' && c <= '9') || (c >= 'A' && c <= 'F') |==> (c >= 'A' && c <= 'F')
1626	||(boolean,boolean):TRUE(boolean,boolean):org.jsoup.parser.CharacterReader@consumeHexSequence:229:(c >= '0' && c <= '9') || (c >= 'A' && c <= 'F') |==> true
1627	>=(int,int):==(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:229:c >= 'a' |==> c == 'a'
1641	<INC/DEC>:<NO-OP>:org.jsoup.parser.CharacterReader@consumeHexSequence:230:pos++ |==> <NO-OP>
1642	-(int,int):%(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:234:pos - start |==> pos % start
1643	-(int,int):*(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:234:pos - start |==> pos * start
1644	-(int,int):+(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:234:pos - start |==> pos + start
1645	-(int,int):/(int,int):org.jsoup.parser.CharacterReader@consumeHexSequence:234:pos - start |==> pos / start

Analysis for each mutant

Mutant Number	Operator Type	What's Changed	Status	Analysis
1600	ROR	`pos < length` changed to `pos != length`	LIVE	it is not equivalent mutant. a test case where `length` is greater that `pos` will kill this mutant.
1601	ROR	`<` changed to `<=`	LIVE	this is not equivalent mutant neither. A test case where `length` is equal to `pos` will kill this mutant.
1602	ROR	`pos < length` changed to `false`	FAIL	mutant killed
1603	ROR	`c >= '0'` changed to `c == '0'`	FAIL	mutant killed
1604	ROR	`c >= '0'` changed to `C > '0'`	FAIL	mutant killed
1605	ROR	`c >= '0'` changed to `True`	LIVE	This is not equivalent mutant. A test case where `c` is less than `0` will kill this mutant.
1606	ROR	`c <= '9'` changed to `C < '9'`	FAIL	mutant killed
1607	ROR	`c <= '9'` changed to `C == '9'`	FAIL	mutant killed
1608	ROR	`c <= '9'` changed to `True`	FAIL	nocomment
1609	COR	`c >= '0' && c <= '9'` changed to `c >= '0' == c <= '9'`	LIVE	This is not equivalent mutant. A test case where `c >= '0'` and `c <= '9'` are both false will kill this mutant.
1610	COR	`c >= '0' && c <= '9'` changed to `false`	FAIL	mutant killed
1611	COR	`c >= '0' && c <= '9'` changed to `c >= '0'`	FAIL	mutant killed
1619	COR	`c >= 'A' && c <= 'F'` changed to `c >= 'A' == c <= 'F'`	LIVE	This is not equivalent mutant. A test case where `c >= 'A'` and `c <= 'F'` are both false will kill this mutant.
1620	COR	`c >= 'A' && c <= 'F'` changed to `false`	FAIL	mutant killed
1621	COR	`c >= 'A' && c <= 'F'` changed to `c >= 'A'`	LIVE	This is not equivalent mutant. A test case where `c >= 'A'` is true but `c <= 'F'` is false will kill this mutant.
1622	COR	`c >= 'A' && c <= 'F'` changed to `c <= 'F'`	FAIL	mutant killed
1623	COR	`(c >= '0' && c <= '9') || (c >= 'A' && c <= 'F')` changed to `(c >= '0' && c <= '9') != (c >= 'A' && c <= 'F')`	LIVE	This is not equivalent mutant. A test case where `(c >= '0' && c <= '9')` and `(c >= 'A' && c <= 'F')` are both true will kill this mutant
1624	COR	`(c >= '0' && c <= '9') || (c >= 'A' && c <= 'F')` changed to `(c >= '0' && c <= '9')`	FAIL	mutant killed

Mutant Number	Operator Type	What's Changed	Status	Analysis
1625	COR	`(c >= '0' && c <= '9') || (c >= 'A' && c <= 'F')` changed to `(c >= 'A' && c <= 'F')`	FAIL	mutant killed
1626	COR	`(c >= '0' && c <= '9') || (c >= 'A' && c <= 'F')` changed to `true`	FAIL	mutant killed
1627	ROR	`c >= 'a'` changed to `c == 'a'`	FAIL	mutant killed
1641	STD	delete operation `pos++;`	TIME	mutant killed
1642	AOR	`pos - start` changed to `pos % start`	LIVE	This is not equivalent mutant. A test case where `case` is more than twice of start value will kill this mutant.
1643	AOR	`pos - start` changed to `pos * start`	EXC	mutant killed
1644	AOR	`pos - start` changed to `pos + start`	EXC	mutant killed
1645	AOR	`pos - start` changed to `pos / start`	FAIL	mutant killed

Here are number of mutants killed and lived: * LIVE = 8 * KILLED = 17

LIVE + KILLED does equal to 25. Here LIVE includes mutants that are not covered and mutants that are covered but not killed. Thus sum equals to total mutant count. Mutation score for this method is KILLED / MUTANTS = 17 / 25 = 68 %. It is in the ballpark of overall mutation score. Although the results do not show coverage information, from the test case and source code, I can see this method is well covered. Effectiveness of the test suite is on the same level of effectiveness of test suite on the project overall.

Operator type referrence:

Operator Type	Description
AOR	Arithmetic operator replacement
STD	Statement Deletion Operator
COR	Conditional Operator Replacement
ROR	Relational Operator Replacement