ESOF 322: Homework 5

River Kelly and Peyton Dorsh

December 2, 2021

Question 1 (25 pts)

a. (10 pts)

1. The system we downloaded was Weka: https://sourceforge.net/projects/weka/.
2. This system uses machine learning algorithms to solve data mining problems.
3. This system has total of 1,871 files in its main java source directory, and a total of 635,973 lines of code.
   To find the total number of lines, we executed to following command:

```
find . -type f -name "*.java" | xargs wc -l
```

b. (15 pts)

1. Capture the output of the tool (without checking the "Search in file content" box) and print it.

```
Found 17 files that possibly contain design patterns.

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\core\CommandlineRunnable.java
Possible patterns: Command

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\core\DictionaryBuilder.java
Possible patterns: Builder

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\core\pmml\PMMLFactory.java
Possible patterns: Factory

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\core\pmml\jaxbbindings\MISSINGVALUESTRATEGY.java
Possible patterns: Strategy

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\core\pmml\jaxbbindings\NOTRUECHILDSTRATEGY.java
Possible patterns: Strategy

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\core\pmml\jaxbbindings\ObjectFactory.java
Possible patterns: Factory

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\experiment\InstanceQueryAdapter.java
Possible patterns: Adapter

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\gui\experiment\GeneratorPropertyIteratorPanel.java
Possible patterns: Iterator

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\gui\knowledgeflow\AbstractGraphicalCommand.java
Possible patterns: Command

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\gui\knowledgeflow\GetPerspectiveNamesGraphicalCommand.java
Possible patterns: Command

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\gui\knowledgeflow\GraphicalEnvironmentCommandHandler.java
Possible patterns: Command

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
    weka\gui\knowledgeflow\KFGraphicalEnvironmentCommandHandler.java
Possible patterns: Command

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
```

```
   weka\gui\knowledgeflow\SendToPerspectiveGraphicalCommand.java
Possible patterns: Command

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
   weka\gui\knowledgeflow\TemplateManager.java
Possible patterns: Template

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
   weka\gui\simplecli\AbstractCommand.java
Possible patterns: Command

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\main\java\
   weka\knowledgeflow\steps\ASSearchStrategy.java
Possible patterns: Strategy

C:\Users\peyto\Documents\CSCI\Fall2021\esof322\WEKA-SRC\src\test\java\
   weka\core\DictionaryBuilderTest.java
Possible patterns: Builder
```

2. How does this tool look for instances of design patterns?

For instance of design patterns, this tool scanned each of the filenames for design pattern conventions. This is because we did NOT select "Search in File Content". Looking at the results of the capture output, you will notice that the Java files that were matched contain the common design patterns naming conventions.

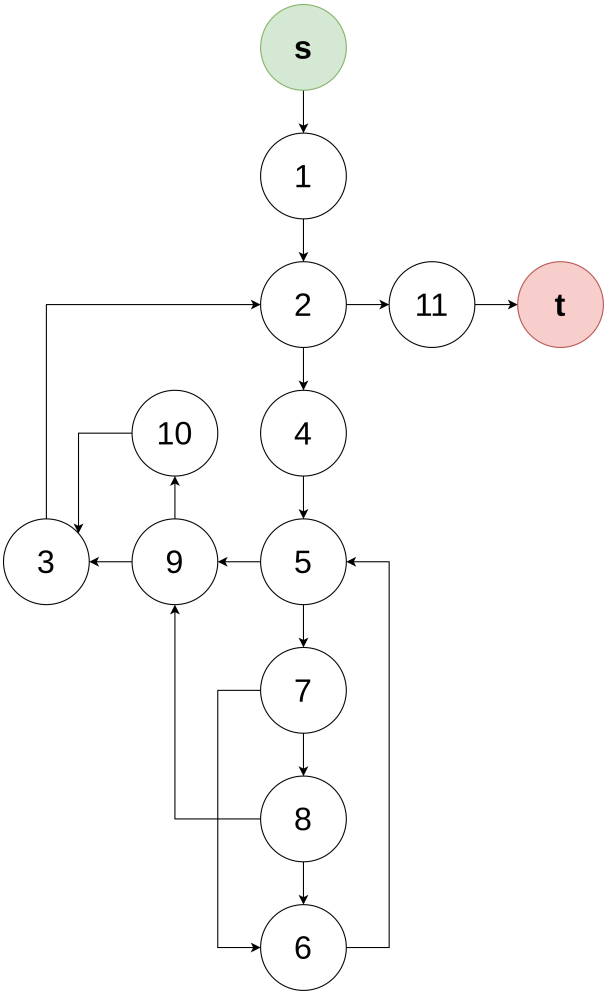
3. Do you think the process used by the tool is correct? How would you do it? Be specific.

We believe the process used by this tool is somewhat accurate. Considering that the tool only matched design pattern files based on common naming conventions, it would be safe to assume that the authors of this open source project consciously named such files with the intention of using a specific design pattern.

To better improve the detection of an observed design patterns, we would have to search the contents of each file. Doing so would provide us a greater understanding of the data model represented within the code. We would be able to detect the association between objects (i.e. files), and examine the routine behavior of how these files interact with one another. Doing so would better present the ability of detecting the use of certain design patterns in the project's code. We would identify a used design pattern based on class inheritance and abstraction.

Question 2 (10 pts)

```
1. /* Find all primes from 2-upper_bound using Sieve of Eratosthanes */
2.
3. #include
4. typedef struct IntList {
5.           int value;
6.           struct IntList *next;
7.1          } *INTLIST, INTCELL;
8. INTLIST sieve ( int upper_bound ) {
9.
10         INTLIST prime_list = NULL;   /* list of primes found */
11         INTLIST cursor;              /* cursor into prime list */
12         int candidate;               /* a candidate prime number */
13         int is_prime;                /* flag: 1=prime, 0=not prime */
14
15          /* try all numbers up to upper_bound */
16         for (candidate=2;
17
18.          2  candidate <= upper_bound;
19.          3  candidate++) {
20.
21.       4 is_prime = 1; /* assume candidate is prime */
22.         for(cursor = prime_list;
23.
24.           5  cursor;
25.           6  cursor = cursor->next) {
26.
27.          7 if (candidate % cursor->value == 0) {
28.
29.            8  /* candidate divisible by prime */
30.               /* in list, can't be prime */
31.               is_prime = 0;
32.               break;  /* "for cursor" loop */
33.             }
34.          }
35.       9 if(is_prime) {
36.
37.         10 /* add candidate to front of list */
38.            cursor = (INTLIST) malloc(sizeof(INTCELL));
39.            cursor->value = candidate;
40.            cursor->next  = prime_list;
41.            prime_list = cursor;
42.          }
43.        }
44.     11 return prime_list;
45.       }
```



1. Test cases that would give 100% Node Coverage (NC)

$T = \{$
$t_1 = \{s, 1, 2, 4, 5, 7, 8, 6, 5, 9, 3, 2, 11, t\},$
$t_2 = \{s, 1, 2, 4, 5, 9, 10, 3, 2, 11, t\}$
$\}$

2. Test cases that would give 100% Edge Coverage (EC)

$T = \{$
$t_1 = \{s, 1, 2, 4, 5, 7, 6, 5, 9, 3, 2, 11, t\},$
$t_2 = \{s, 1, 2, 4, 5, 7, 8, 6, 5, 9, 3, 2, 11, t\},$
$t_3 = \{s, 1, 2, 4, 5, 7, 8, 9, 3, 2, 11, t\},$
$t_4 = \{s, 1, 2, 4, 5, 9, 10, 3, 2, 11, t\}$
$\}$

3. Is 100% NC or 100% EC possible in general? Why, or why not?

Generally speaking, 100% node coverage and 100% edge coverage is possible, but speaking practically that is not always the case. Because the set of tests is finite for a given domain, brute force tells us that we could exhaust all potential circumstances. But, the first thing you learn after learning brute force, is that it is typically the worst way of doing things. As computer scientists, we strive for efficiency. In terms of the set of "real-world" tests, this would imply that we provide test cases which are practical, not absolutely encompassing.

Question 3 (10 pts)

The application we chose to test was YouTube (https://www.youtube.com/). The scope of our test was videos related to reading sheet music. In out Metamorphic Testing, we began by generating a initial test (base-case) using the search string "*read sheet music*". Then, for each of our Metamorphic Relations we slightly altered the query string and compared the results.

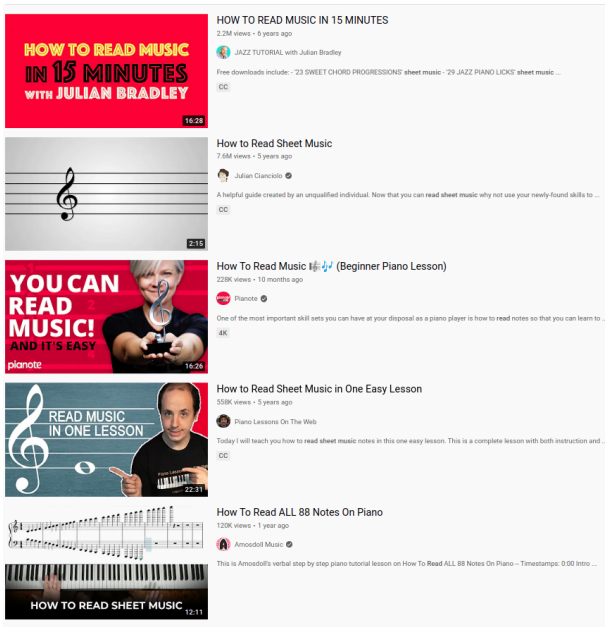
We hypothesize that when searching for a video you YouTube, a user will most likely click on one of the videos suggested on the initial preview (i.e. without scrolling down). To classify a Metamorphic Relation in our test as passing, we required that at least 2 videos shown in the initial test must be shown in the MR to pass, otherwise the MR test was considered a failure.

MR tests preview

1. **MR #1**: pass - Query String: *"how to read sheet music"*
2. **MR #2**: fail - Query String: *"reading sheet music"*
3. **MR #3**: pass - Query String: *"read" "sheet" "music"*

Initial Test

The query string used: "*read sheet music*"

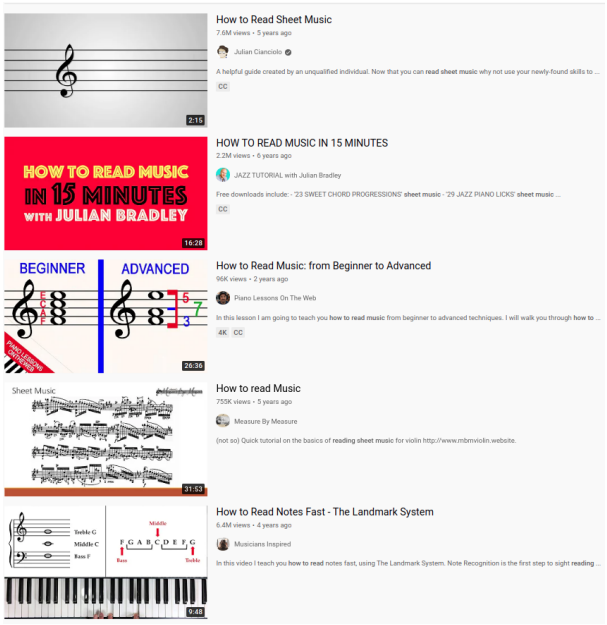


Metamorphic Relations

MR#1

Justification: We justified that adding the string "how to" before "read sheet music" should be implied implicity considering the scope of our test (i.e. a user wishing to learn 'how to' read sheet music).

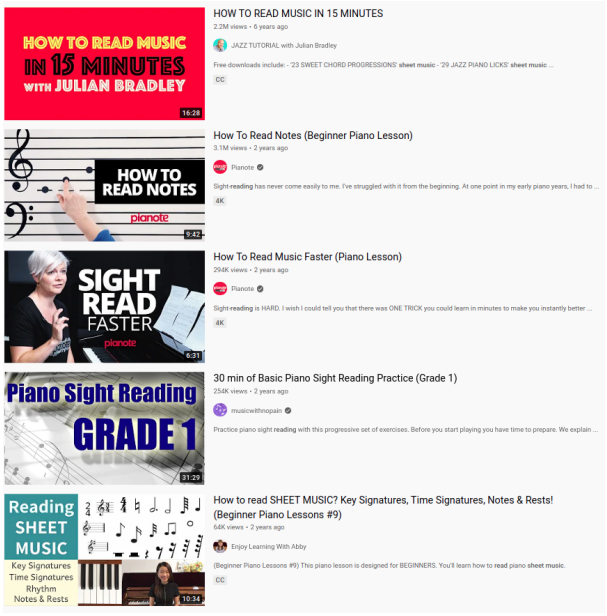
Query String: "*how to read sheet music*"



MR#2

Justification: From our original phrase, we changed the verbal tense of the word "read" to "reading". We concluded that this modification should not alter the search results.

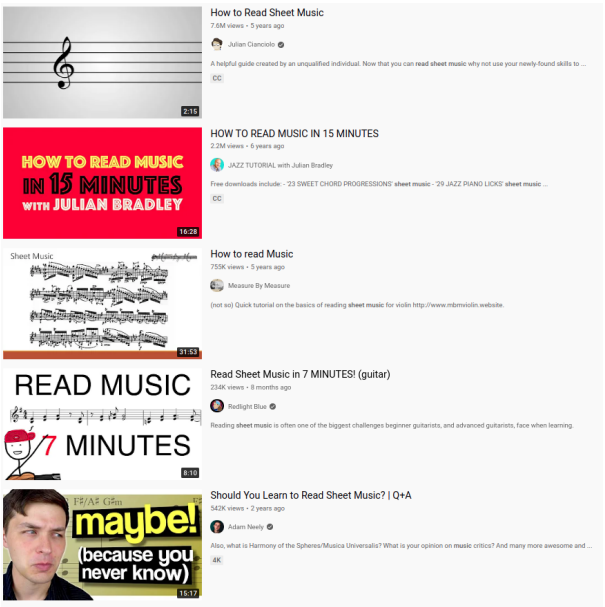
Query String: *"reading sheet music"*



MR#3

Justification: We added quotes around each of the words from the original phrase.

Query String: *"read" "sheet" "music"*



Question 4 (12 pts)

1. Line 6: if $(i < 1)$
   Test case $i = 1$ will kill this mutant.

2. Line 6: if $(i == 1)$
   Test case $i = 0$ will kill this mutant.

3. Line 12: $fib2 = fib$;
   Test case $i = 3$ will kill this mutant.