DLSU Research Congress 2021
De La Salle University, Manila, Philippines
July 7 to 9, 2021

Development of a Local Code Similarity Checker with Winnowing Algorithm in JavaFX using Software Engineering Techniques

Reginald Geoffrey L. Bayeta IV[1], Isaiah Jassen L. Tupal[1], and Melvin K. Cabatuan[1]
[1] *Department of Electronics and Communications Engineering*
**isaiah_tupal@dlsu.edu.ph*

Abstract: This paper presents the design and development of a local code similarity and measurement system in JavaFX platform, using a popular algorithm utilized to fingerprint documents, the Winnowing algorithm, in which, fingerprints were then compared to each other using the dice similarity coefficient as the definition of similarity. The final application was developed using software engineering techniques, which includes software requirement specification (SRS), user stories, use-case diagram, object and data flow diagrams, and elementary software metrics.
Key Words: winnowing, software similarity, javafx, software metrics, dice

1. INTRODUCTION

De La Salle University imposes strict consequences on academic dishonesty, especially during the online learning setting. Actions that fall under this term include plagiarism and copying one's output as their own.

For programming classes in engineering and computer studies colleges, academic dishonesty can be done through direct source code copying and code selling. Copying the source code from someone else is considered as an act of academic dishonesty because it is expected that students in programming classes should solve problems on their own. One way to determine if the source code is copied is to check its similarity with the suspected original source code.

Using code similarity to detect academic dishonesty, specifically plagiarism, is not new. Systems to detect software similarity have been in place for at least since the 80s

Micheal Rabin and Richard Karp developed an algorithm (Leman, Rahman, Ikorasaki, 2019) to find string patterns called the Rabin-Karp Algorithm. The idea behind this is the use of a hashing function to convert a string into value that is easier to match.

For Alex Aiken's MOSS, or Measure of Software Similarity, is a system that measures how similar programs written in C, C++, Java, Pascal and other programming languages are from each other. (Bowyer & Hall,1999) The MOSS was developed in 1994. The theory behind MOSS is the winnowing algorithm. Winnowing is similar to the Rabin-Karp algorithm in which both uses hashing as an important aspect of it.

The program developed for this paper utilizes this algorithm as its main fingerprinting algorithm. There are many reasons for implementing winnowing for this project. The first reason is that this algorithm has been tried and tested as it is being utilized in MOSS, a respected and widely used system. Another reason is that it exhibits desirable three properties that are important for measuring similarity: Insensitivity, Noise suppression, and Position Independence.

Insensitivity

The algorithm should not be affected by extra whitespace and newlines or by capitalization and punctuation

(a) This sentence is a match
(b) Thissentenceisamatch
(a) and (b) should be a match

Fig. 1 Example of insensitivity

2. METHODOLOGY

2.1 Software Requirement Specification (SRS)

External Interface Requirement

1. User interface
   The software includes a graphical user interface to be utilized by the user.
2. Hardware interface
   The software requires an operating system that has a *java* compiler to be used.
3. Software interface
   The software is written in *Java* programming language employing the *JavaFX* platform.

Functional Requirement

1. Directory reading
   The software shall accept a directory as an input wherein each file in the directory will be accepted as a data entry.
2. File concatenation
   Given the input directory, the software will traverse each data entry to read text editable file format, more specifically *.cpp* and *.java* files.
3. Similarity matrix generator
   For a given data entry, it will be compared to each entry in all submissions including itself wherein the output will be a matrix of the software similarity scores. Similarity score will be a value from 0.0 to 1.0.
4. Program metrics generator
   For each data entry, program metrics will be determined by the program and will be outputted as a matrix.
5. Matrix and metrics exporter
   The user can export the resulting matrix and metrics into an *.xlsx* file for whatever purpose it may serve.

2.2 User Stories

In designing the system, a user story from the client must be depicted. Software idea usually comes from the needs or wants of a user or consumer. User stories generally refers to a short description about the features of the programs, who is it for, and the reason why it is needed (Dalpiaz & Brinkkemper, 2018). For this project, the authors decided to depict the client as a professor or instructor of a programming class with the following user story:

1. View the list of submission files for a given deliverables.
2. View the submission of a given student in search in one place.
3. View the similarity percentage between two students' submissions.
4. View a similarity percentage matrix for students' submission for a given deliverable.
5. Export the corresponding results to an .xlsx file.

Since a user-story is just like a gist of the core requirements, several changes are observed in the development of the system. During the development stage, some functionalities stated in the initial requirements have been changed to comply with the specifications stated in the SRS.

2.3 Use-Case Diagram

In the designing phase of the system, a use-case diagram was created to model the behavior of the core variable and functionalities within a given system. Figure 2 shows the initial requirements depicted from the user-stories wherein it shows the interaction between a professor and a student. Both professor and students can login, view submission files, and view similarity score of the submission but the professor is the one responsible for creating an assignment where the students can submit submissions to.

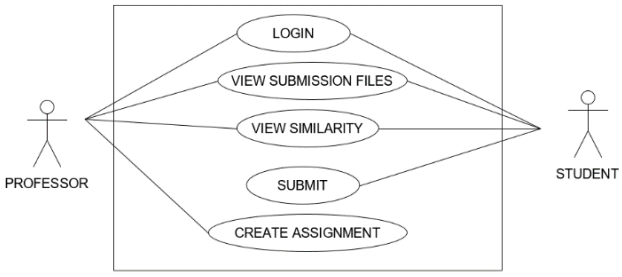

Fig. 2 Use-case diagram

Due to several constraints and the additional requirements introduced later in the development phase, the login, submit, and create assignment functions were not included in the final version of the program.

2.4 Modular Architecture

Every project requires the design of modular structure since it allows a problem to be separated into modules and recombines the solution to make the workflow more efficient (Pinto 2019). As seen in Figure 3, the system is mainly divided into following five main

modules:

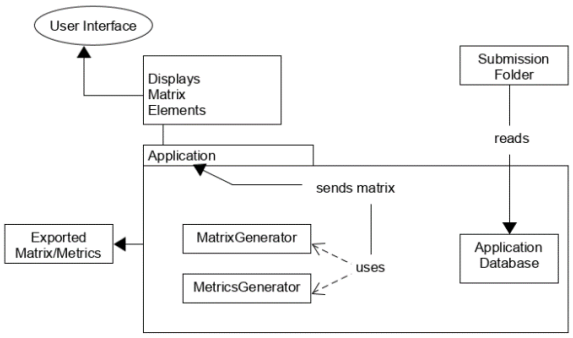

Fig. 3 The system modular architecture
(1) Graphical User Interface (GUI) component, (2) File handling component, (3) Similarity matrix generator, (4) Software metrics generator, and (5) Matrix/Metrics Exporter.

2.5 Object Diagram

Figure 4 presents the object diagram of the system. The main objects used in the software include MatrixData, MetricsData, Top10Data, and Exporter. Names of the object may differ from the main source code for simplification purposes. Object diagram is used to show the partial view of the structure of the software.

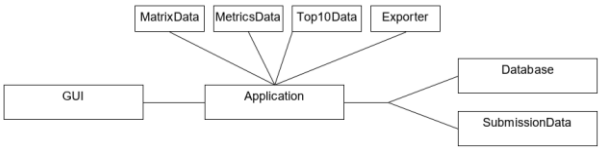

Fig. 4 The system object diagram

2.6 Data Flow Diagram

Figure 5 highlights the flow of execution of the software functionalities. The user must choose a valid submission directory wherein the program will parse *.java* and *.cpp* files wherein each submission entry will be read in both per line and per word basis which will be stored in their respective array list. From this, the program shall generate matrix and program metrics statistics upon requests while generating the Top 10 submissions with the highest similarity score. Upon viewing desired output, the user can choose to save these results by exporting it to an *.xlsx* file.

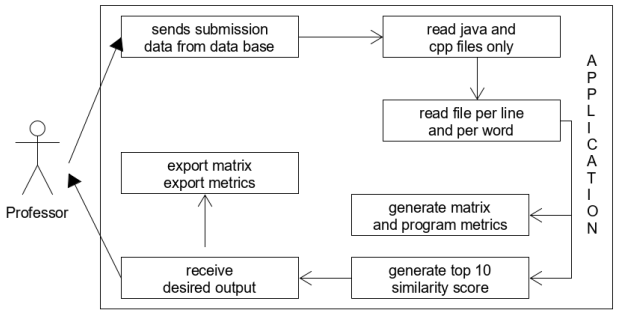

Fig. 5 The system data flow diagram

Similary Matrix and Program Metrics Generator
In this section, the algorithms behind each core functionalities of the program will be discussed.

2.7 Algorithms and Metrics

The very heart of program similarity is the winnowing algorithm. To process a document or source code using winnowing, it is first preprocessed. The file is first tokenized. Irrelevant features such as comments, dots, capitalization, newlines and whitespaces are removed. The tokens are then concatenated into a single string. K grams were then derived from this string and these k grams are then hashed and put into a set. Another set of k gram were then derived from the set of hashes. These k grams hashes are the potential fingerprints of this document

Algorithm 1: String to k-grams hashes

1. Input: stringCode, K
2. Output: listofKgrams
3. hashesSet=emptySet()
4. For i = 0 to stringCode.len
5. Temp = “”
6. //resets the temp into an empty string
7. For j = i to i+k
8. Temp.concat(stringCode[]
9. hashesSet.add(hash(temp))
10. Return hashesSet

These k grams of hashes are fed into the winnowing algorithm. The idea behind winnowing is that there are parameters K and T that determines how the fingerprinting will process. K is the *noise threshold* in which matching strings with length of less than k will be not detected. T is the guarantee threshold in which matches of strings with length greater than T will be detected. Using these two parameters, another parameter W (window) can be computed. W is T-K+1. The K used in preprocessing is the same K all

throughout the fingerprinting process. The pseudocode for the winnowing algorithm implemented in this project is written below:

Algorithm 2: Winnowing algorithm

```
1.  Input: h t, k
2.  w = t-k+1
3.  min = maxValue , minIndex=-1
4.  // maxValue is the largest integer value stored
5.  fingerPrints = emptySet()
6.  for i = 0 to h.size
7.    if minIndex == i-1
8.       min = maxValue
9.       for x = i to w+1
10.         if min > h.get(x % h.size)
11.            minIndex= x % h.size
12.         min = h[( x % h.size]
13.      fingerPrints.add(min)
14.   else
15.      if min>h[(i+w)%h.size)]
16.         min = h[ (i+1) %  h .size]
17.         minIndex = (i+w) % h.size
18.         fingerPrints.add(min)
19. return fingerPrints
```

The idea behind winnowing is that given a hashes of k grams, the program picks the minimum hash in each window. The window is the subset of the set of hashes that the program views in each cycle. The window size W is determined by guarantee threshold K and by the noise threshold T. A hash can only be recorded once so if hash is previously recorded in a window before the current one, it will not be recorded again even if it is the smallest hash in the current window. These hashes will serve as the document's fingerprint.

The fingerprints derived from this process are then used as the set for the dice similarity coefficient. (Wibowo, Sudarmadi, & Barmawi, 2013) The dice similarity coefficient is defined as follows:

$$DC = 2\,\frac{|A \cap B|}{|A| + |B|}$$

Fig. 6 dice coefficient formula

The A and the B in this formula is the set obtained from winnowing which is the fingerprint of a particular document. The value shown in the similarity matrix is the Dice coefficient of the two projects

In order to get a given program metrics, we need to get the count of its unique and total operators and operands. Program metrics fetched in this paper are program difficulty, vocabulary, length, level, volume, and effort. The following are the formulas used to get each metric:

1. Program vocabulary, n
   $n = n_1 + n_2$
   n_1 = number of unique operators
   n_2 = number of unique operands
2. Program length, N
   $N = N_1 + N_2$
   N_1 = total occurrences of operators
   N_2 = total occurrences of operands
3. Program volume, V
   $V = N * \log_2 n$
4. Program level, L
   $L = 2n_2 / (n_1 N_2)$
5. Program difficulty, D
   $D = 1/L$
6. Program effort, E
   $E = D * V$

In essence, in order to get the values of the given program metrics, we need to have the list of all operators and operands found in the program. Algorithm 3 shows the process of getting such parameters used to generate the program metrics of a given program submission.

Algorithm 3. Program Metrics Generator
Input: ArrayList lineList
ArrayList operatorMainList
Output: ArrayList operatorsList, operandsList

lineList – this is the list of lines per code read by the program for each submission entry
operatorMainList – list of operators expected to be read
by the program (read from a text file)
operatorsList – list that will contain all operators found
per submission entry
operandsList – list that will contain all operands found per
submission entry

1. **boolean** isComment = false
2. **for** String line in lineList **do**

In lines 3 – 10, the program will ignore the line that contains a comment with the following format:

matchCount(String line, String operator) : int – this function returns the occurrence of an operator in each line.

```
3.  if matchCount(line, "//") ! = 0 do
4.  line = line.substring(0, line.indexOf('/')-1)
5.  end if
6.  if matchCount(line,"/*") != 0 && matchCount(line, "*/") != 0) do continue
7.  if matchCount(line, "/*") != 0 do
8.  isComment = true, continue
9.  if matchCount(line, "*/") != 0 do continue
10. else isComment = false
11. for String pattern in operatorMainList do
12. int a = matchCount(line,pattern)
13. if a != 0 do
14. for int j = 1 to a do
15. operatorsList.add(pattern)
16. end for
17. end if
18. end for
```

In this algorithm, anything that is not an operator in considered an operand except for comments.

```
19. String[] operands = line.split(" ")
20. for String words in operands do
21. words = words.replaceAll("\\W", " ")
22. if words != "" | | words != " " do
23. operandsList.add(words)
24. end if
25. end for
26. end for
```

3. RESULTS AND DISCUSSION

Obtained Data Sets

This project is part of the requirements for the course Software Design and Engineering in De La Salle University's Computer Engineering program. Data sets used in this paper are all source code of fellow students who also developed software with the more or less same software requirements. Each data entry is downloaded from their respective *github repositories* by utilizing a *python script*.

Software Similarity Score Matrix

After reading and filtering each data entry in the data set a similarity score matrix will be displayed once requested. Figure 8 displays the resulting matrix of the obtained final data set. Once the similarity score matrix is displayed, the Top 10 data entries with the highest obtained score will be also showed.

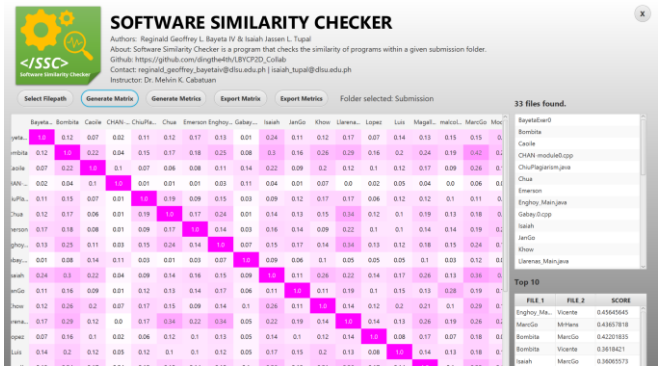

Fig. 8 Sample similarity score matrix

As seen in the GUI, there is a black diagonal. This is because we set darker colors as more similar and the diagonal is basically the same file being compared to each other.

Program Metrics Matrix

Similarly, after the file reading, the program will show the program metrics matrix upon request of the user. Figure 13 highlights the resulting program metrics of the obtained data set. Program metric attributes include program vocabulary, length, volume, level, difficulty, and effort.

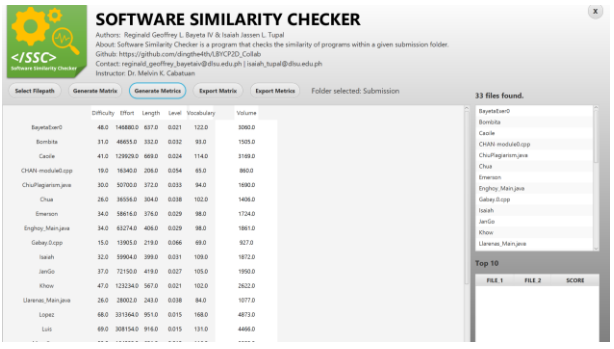

Fig. 9 Sample program metrics matrix

Similarity Matrix and Program Metrics Exporter

Upon request, the user can opt to export the resulting similarity matrix and program metrics into an *.xlsx* file. Figure 15 shows the sample exported matrix. An exported version helps to access results without having to run the program and conduct tests again.

Name	Difficulty	Effort	Length	level	vocabular	volume
LBYCP2D_	48	146880	637	0.021	0.021	122
LBYCP2D_	69	308154	916	0.015	0.015	131
LBYCP2D_	53	164035	651	0.019	0.019	116
LBYCP2D_	27	36234	309	0.038	0.038	77
LBYCP2D_	40	63920	357	0.025	0.025	88
LBYCP2D_	39	100659	558	0.026	0.026	102
LBYCP2D_	19	14839	187	0.052	0.052	65

Fig. 10 Sample exported matrix

Unit Testing

After finishing the exporter, the last feature of the program, several key algorithms were then tested via Unit testing or by timing the speed of the program. The Functions needed to measure the similarity were unit tested, Fig. 11 .

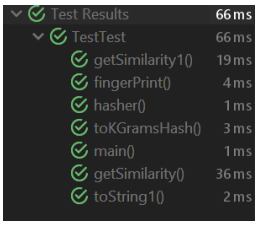

Fig. 11 Unit testing

4. CONCLUSIONS

This study illustrated the development and implementation of JavaFX software as the GUI platform for implementing code similarity using winnowing for fingerprinting, and utilizing software engineering techniques. Additionally, the researchers illustrated the automated measurement of software metrics as an added feature to the developed application. Winnowing is known for noise suppression and insensitivity, thus, the scores are generally small as shown in the results.

5. REFERENCES

Dalpiaz, F., & Brinkkemper, S. (2018). Agile Requirements Engineering with User Stories. *2018 IEEE 26th International Requirements Engineering Conference (RE)*. doi: 10.1109/re.2018.00075

K Bowyer, W., and Hall, L. O. (1999). Experience using 'MOSS' to detect cheating on programming assignments. In: 29th ASEE/IEEE Frontiers of Education Conference, San Juan, Puerto Rico, pp. 18--22.

Leman, D., Rahman, M., Ikorasaki, F., Riza, B. S., & Akbbar, M. B. (2019). Rabin Karp And Winnowing Algorithm For Statistics Of Text Document Plagiarism Detection. 2019 7th International Conference on Cyber and IT Service Management (CITSM). doi: 10.1109/citsm47753.2019.8965422

Wibowo, A. T., Sudarmadi, K. W., & Barmawi, A. M. (2013). Comparison between fingerprint and winnowing algorithm to detect plagiarism fraud on Bahasa Indonesia documents. *2013 International Conference of Information and Communication Technology (ICoICT)*. doi: 10.1109/icoict.2013.6574560