9th International scientific conference
**Technics and Informatics in Education – TIE 2022**
16-18 September 2022

Session: *Engineering Education and Practice*
Professional paper
DOI: 10.46793/TIE22.177J

# Determining source code repetitiveness on various types of programming assignments

Željko Jovanović[1*], Mihailo Knežević[1], Uroš Pešović[1], Slađana Đurašević[1]
[1] University of Kragujevac, Faculty of technical sciences Čačak, Serbia
[*] zeljko.jovanovic@ftn.kg.ac.rs

**Abstract:** Software projects code duplication and plagiarism are very important in various test cases. The purpose of the work presented in this paper is to observe how various software architectures, project structures, and coding approaches generate different views on code changes. In this paper, code plagiarism - code comparing, in different types of projects has been analyzed through two different approaches. Python script based on the sequence matcher function and the GitLab compare tool are analyzed and compared. Results are presented and discussed in the paper.

**Keywords:** code repetitiveness, duplicate code detection, python, GitLab compare, web application

## 1. INTRODUCTION

It is widely believed that software projects have certain similarities to each other. Similarities in programming imply similarities in their solutions.

According to that, it is quite obvious that copying code from someone else happens very often [1]. After copying solution-specific code, it has to be adjusted in order to be reused in some other project. This could be done in potentially similar proposed features but usually with different project design concepts and architecture. In some broader sense, this means that new software products are based on older code [2], [3]. In some corner cases even on reverse-engineered code. It has been noticed that for some high confidentiality source code, methods such as code obfuscation can protect the final product from reverse engineering.

Besides its vast importance in the software development industry, code plagiarism detection plays a significant role in machine learning and deep learning research efforts as identifying repetitive pieces of code can lead to making any future progress in code writing automation.

Also, it is worth mentioning that code plagiarism detection methods are necessary for cheat-proofing programming assignments in engineering universities and schools throughout the world [4], [5], [6].

There are several code plagiarism tools used for this purpose nowadays, such as Codeleaks, Codequiry, Codegrade, Moss, and Unicheck. Almost all of those tools use the benefits of AI pattern recognizing capabilities and as such require quite a lot of computing power. On the other hand, these tools are not free of charge and as such are not fitting into the philosophy of this work which aims to analyze as simple as possible ways of detecting code plagiarism. The authors in this paper attempt to test new tools and functions by avoiding standard, commercial solutions.

Besides these, there are some free tools in the form of desktop apps and web online solutions like WinMerge, CodeCompare, and Diffchecker. Even if they could do the purpose, the focus was to use tools that are learned during studies in faculty and try to extend their usage to some new purposes.

In this paper, Python script and GitLab compare functionality are analyzed for the aim of laying the foundations for the development of a new system that would be used for these purposes.

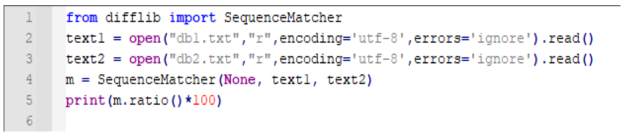
The paper is structured as follows: at first, the used methods are explained. After that, three different test cases of code samples and project structures are presented. The paper finishes with results, conclusions, and ideas for future work.

## 2. USED METHODS

In all three cases, analysis has been done using two methods: the modified integrated GitLab compare tool and the Python script provided in Fig. 1.

The first method is based on the integrated Gitlab compare tool. In order to use the GitLab compare tool properly, source code files whose differences we seek to find should be put into different commits on different branches. After that integrated comparator can be used to compare code files line by line, thus producing differences between two files which is suitable for version control systems and software project progress tracking needs.

The second method is based on the Python script which uses *difflib* [7] library and a *SequenceMatcher* [8] function. Difflib library contains classes and functions for comparing sequences. It can be used for example, for comparing files, and can produce information about file differences in various formats ranging from text matching (which is our case) up to image comparison [9]. SequenceMatcher is part of the difflib library which covers the task of finding code similarities on the character level. SequenceMatcher leverages Ratcliff/Obershelp pattern recognition (also known as Gestalt pattern matching) [10] and code comparison using such method produces detailed and qualitatively stable comparison and  as such is very suitable for the required purpose.



```python
from difflib import SequenceMatcher
text1 = open("db1.txt","r",encoding='utf-8',errors='ignore').read()
text2 = open("db2.txt","r",encoding='utf-8',errors='ignore').read()
m = SequenceMatcher(None, text1, text2)
print(m.ratio()*100)
```

**Figure 1.** *Python script used for comparing code files*

In contrast to GitLab compare results, the output of Python script is the percentage of similarities/duplication of two code files determined by *SequenceMatcher* imported from *difflib* library.

## 3. TEST CASES

In this paper, the repetitiveness of programming code has been analyzed in three test cases which are very different. Different programming languages, code, and project structures between test cases are used. Three specific cases have been covered.

### 3.1. First Test Case - Change Of A Single Line Of Code In A Boilerplate (Prepared For Reusability) Code

Observed code is a connection file that connects a database with an application. Code is written in the PHP programming language and is used as boilerplate code. It defines parameters for PDO (PHP Data Objects) like hostname, port, username, and password for MySQL server connection. It is expected to be involved in all projects that use PDO connections to MySQL databases. Before and after modification code contains 29 lines of code. The expected output of the comparator function should be very high.

### 3.2. Second Test Case - Solution Of The Same Task In The C Programming Language, With And Without Using Functions.

In both cases, the code solves the basic programming assignment of entering and printing out array elements. If solved without functions, source code is 33 lines long as opposed to 48 lines of code for a solution with functions. In this case, code matching in some percent should be detected even if there was no plagiarism between authors since the two approaches are applied to solving the same problem.  Solutions are very different in structure, but still similar in a textual manner.

### 3.3. Third Test Case - Four Different Implementations of a Large-Scale Web Project

Projects are created as practical work within the "Internet programming" course exam at the Faculty of technical sciences Cacak. The course is scheduled in the VIII semester (IV year) as one of the final courses before graduation. It relies on the acquired knowledge from several other courses so a large variety of techniques, platforms, and software architecture patterns could be used. The subject of the practical work was to develop a dynamic Web site for *recreational tennis* using PHP and JS programming languages with a responsive front user interface design. It consists of 19 functional tasks (presented in Table 1) which could be developed in any desired way so that the functional requirements are met. Four separate teams were created and they had daily and weekly scrum meetings (what is done and what should be done in the project for every individual team member) within the team.  In this case, not all four project implementations have covered all 19 feature requirements. Details of covered features per team are provided in Table 1. Since all implementations have only 7 out of 19 features in common (about 37% of all features) and taking into consideration that all implementations have completely different approaches, a high percentage of code matching was not expected since it would lead to code plagiarism between teams. It is unnecessary to emphasize that the total program lines of code for these projects are quite large: team 1 has a total of 11462 lines of code, team 2 has 4171, team 3 sums up to 9009 lines of code while team 4 has a total of 7905 program code lines.

**Table 1.** *Large scale web application project features*

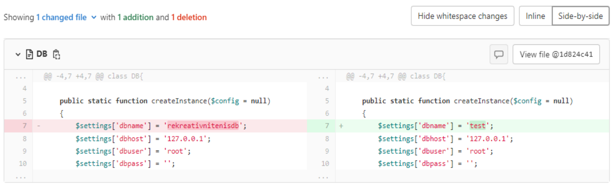
| N | FEATURE | T1 | T2 | T3 | T4 |
|---|---|---|---|---|---|
| 1 | Log of played matches between recreational players and record of results are to be taken care of by application | + | + | + | + |
| 2 | Players and clubs can register and edit their profiles | + | + | + | + |
| 3 | Clubs can register and edit their court profiles. | + | | | + |
| 4 | Players and clubs can login with valid credentials (email, password) | + | + | + | + |
| 5 | Matches can be filtered (filter example: list all yesterday/today/tomorrow matches) | + | | + | + |
| 6 | Players and clubs can reserve matches and keep match log | + | + | + | + |
| 7 | Player/Club can perform court availability check | + | | | + |
| 8 | Auto-fill of required fills while creating a new match based on who is logged in | + | + | | + |
| 9 | Admin (insert score, ban player, delete match, ban club…) | + | + | + | + |
| 10 | Photos upload | + | + | + | + |
| 11 | Support for doubles matches | + | | | |
| 12 | Player ranking based on Wins/Losses ratio | + | + | | + |
| 13 | User profile edit | + | + | + | + |
| 14 | Player ranking with filtering(filter examples: current week, last week, this month, this year) | | + | | + |
| 15 | Create a new tournament (name, description, place) | | | | |
| 16 | Scheduling matches | | | | |
| 17 | Activity information (example: scheduled match confirmation sent by email) | | + | + | |
| 18 | Favorite clubs, adding club to list of favorites | + | | | |
| 19 | Favorite players, add a player to list of favorites | + | | | + |

## 4. RESULTS

In this section results obtained by GitLab compare and Python script in all three test cases will be presented.

### 4.1. First Test Case Results

GitLab compare tool. The code differs in only one line of code, and it is shown in Fig. 2.



```
@@ -4,7 +4,7 @@ class DB{

    public static function createInstance($config = null)
    {
-       $settings['dbname'] = 'rekreativnitenisdb';
        $settings['dbhost'] = '127.0.0.1';
        $settings['dbuser'] = 'root';
        $settings['dbpass'] = '';
```

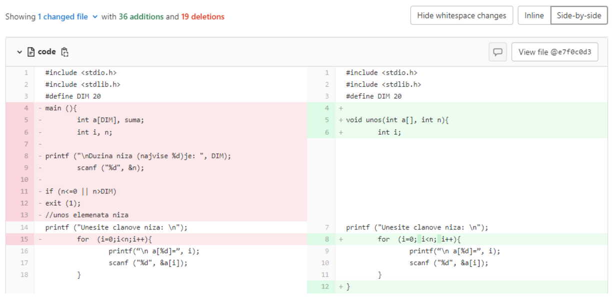
```
@@ -4,7 +4,7 @@ class DB{

    public static function createInstance($config = null)
    {
+       $settings['dbname'] = 'test';
        $settings['dbhost'] = '127.0.0.1';
        $settings['dbuser'] = 'root';
        $settings['dbpass'] = '';
```

**Figure 2.** *Diff image of database connection file provided from compare function in Gitlab*

Python script. Thus, the two codes are quite similar which is algorithmically confirmed by getting a 98.84% matching percentage.

### 4.2. Second Test Case Results

GitLab compare tool. As mentioned above, solutions with and without functions will differ greatly in structure, so diff images generated from GitLab will show that the two source codes are quite different. For practical reasons, only part of the diff image is provided in Fig. 3.



```
  #include <stdio.h>
  #include <stdlib.h>
  #define DIM 20
- main (){
-         int a[DIM], suma;
-         int i, n;
-
- printf ("\nDuzina niza (najvise %d)je: ", DIM);
-         scanf ("%d", &n);
-
- if (n<=0 || n>DIM)
- exit (1);
- //unos elemenata niza
  printf ("Unesite clanove niza: \n");
-         for  (i=0;i<n;i++){
                  printf("\n a[%d]=", i);
                  scanf ("%d", &a[i]);
          }
```

```
  #include <stdio.h>
  #include <stdlib.h>
  #define DIM 20
+
+ void unos(int a[], int n){
+         int i;
  printf ("Unesite clanove niza: \n");
+         for  (i=0; i<n; i++){
                  printf("\n a[%d]=", i);
                  scanf ("%d", &a[i]);
          }
+ }
```

**Figure 3.** *Diff image of C programming assignment provided from compare function in Gitlab*

Python script. On the other hand, the matching percentage determined by the Python script is 37% which proves that the two codes, although structurally different, indeed have a shared code base.

### 4.3. First Test Case Results

GitLab compare tool. In this case, since files contain thousands of lines of code, diff image would be too impractical to be provided here. As an effective alternative GitLab compare Addition/Deletion output (numerical indicator on how many lines of code are Added/Deleted) will be provided. In the same table, a number of mutual lines of code and percent values of duplication/plagiarism will be provided as well.

**Table 2.** *GitLab compare statistics for Test case 3*

| GitLab Compare Addition/ Deletion | Team 1 | Team 2 | Team 3 | Team 4 |
|---|---|---|---|---|
| Team 1 | 11462 | **505** 4.4% 12.2% | **880** 7.7% 9.8% | **1232** 10.7% 15.6% |
| Team 2 | 3666/ 10957 | 4171 | **678** 16.2% 7.5% | **657** 15.7% 8.3% |
| Team 3 | 8129/ 10582 | 8331/ 3493 | 9009 | **1015** 11.3% 12.8% |
| Team 4 | 6673/ 10230 | 7248/ 3514 | 6890/ 7994 | 7905 |

Data in Table 2 are organized as follows. The main diagonal contains the code line number per team. In the lower triangle of the above-mentioned table, the number of Added/Deleted code lines is provided.
In the upper triangle, are the main results of the comparison and it consists of 3 values.

- The upper value is the number of mutual lines of code detected. A number of mutual lines of code are calculated either by subtracting the number of added lines of code from the target code lines number or by subtracting the number of deleted lines of code from the source number of code lines.
- The middle value is the percentage value of detected code in the first-column team code
- The bottom value is the percentage value of detected code in the first-row team code

For example, Team 1 vs Team 2 comparison detected **505** duplicate lines of code which are **4.4%** of Team 1 code, and **12.2%** of Team 2 code.

Presented results vary from the lowest 4.4% to the highest 16.2% of code duplication between teams. Since the Web application project is analyzed, which contains some boilerplate code that has to be the same in all teams, the presented results show that there was no plagiarism between teams.

Python script. Since there are four independent source codes, their matching to each other is provided in the table.

**Table 3.** *Percentage of code match between four projects*

| Matching of Code (%) | Team 1 | Team 2 | Team 3 | Team 4 |
|---|---|---|---|---|
| Team 1 | | 1.08 | 0.64 | 0.76 |
| Team 2 | 1.08 | | 2.28 | 3.11 |
| Team 3 | 0.64 | 2.28 | | 1.95 |
| Team 4 | 0.76 | 3.11 | 1.95 | |

As expected, since projects have been implemented in quite different ways, the matching percentage is low.

Calculated results by both methods are confirmed at the project presentation where all four teams presented completely different solutions both visually and functionally.

## 5. CONCLUSION

From previous results, the conclusion regarding the usability of various methods of comparison to different sizes of source code files. Shortcode files can be easily compared by either the *SequenceMatcher* function or the *GitLab* compare tool. On the other hand, big source code files are very difficult to compare using the diff Gitlab function, so rough code difference estimation should be done using analytical methods. It is worth mentioning here that big source codes can be compared using diff in the Gitlab method as well, but navigating through code and its differences gets very difficult. Using diff Gitlab creates the great benefit of exactly knowing what code changes have been applied, and as such is a valuable tool for the Version Control System.

For future work, frontend and backend files in large-scale web applications would be analyzed separately. Different user interface designs could be based on the same backend code, as well as one user interface design could be used for different backend logic implementations. Also, different functions and tools would be tested for these purposes.

For automation of plagiarism detection, maximum acceptable values should be determined according to project type and assignments.

**ACKNOWLEDGEMENT**

This study was supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, and these results are parts of the Grant No. 451-03-68/2022-14/200132 with the University of Kragujevac - Faculty of Technical Sciences Čačak.

## REFERENCES

[1] Ragkhitwetsagul, C., Krinke, J. & Clark, D. A comparison of code similarity analysers. *Empir Software Eng* 23, 2464–2519 (2018). https://doi.org/10.1007/s10664-017-9564-

[2] Ganguly, D., Jones, G.J.F., Ramírez-de-la-Cruz, A. *et al.* (2018), Retrieving and classifying instances of source code plagiarism. *Inf Retrieval J* 21, 1–23. https://doi.org/10.1007/s10791-017-

[3] Yamamoto, T., Matsushita, M., Kamiya, T., & Inoue, K. (2005, June). Measuring similarity of large software systems based on source code correspondence. In *International Conference on Product Focused Software Process Improvement* (pp. 530-544). Springer, Berlin, Heidelberg.

[4] Oscar Karnalim. (2021) Source code plagiarism detection with low-level structural representation and information retrieval. *International Journal of Computers and Applications* 43:6, pages 566-576.

[5] Novak, M., Joy, M., & Kermek, D. (2019). Source-code similarity detection and detection tools used in academia: a systematic review. *ACM Transactions on Computing Education (TOCE)*, *19*(3), 1-37.

[6] Ahadi, A., & Mathieson, L. (2019, January). A comparison of three popular source code similarity tools for detecting student plagiarism. In *Proceedings of the Twenty-First Australasian Computing Education Conference* (pp. 112-117).

[7] Helpers for computing deltas, Python 3.10.5 documentation, Last access: 20.07.2022, https://docs.python.org/3/library/difflib.html

[8] SequenceMatcher Objects, Helpers for computing deltas, Python 3.10.5 documentation, Last access: 20.07.2022, https://docs.python.org/3/library/difflib.html#difflib.SequenceMatcher

[9] C. Ragkhitwetsagul, J. Krinke and B. Marnette, "A picture is worth a thousand words: Code clone detection based on image similarity," *2018 IEEE 12th International Workshop on Software Clones (IWSC)*, 2018, pp. 44-50, doi: 10.1109/IWSC.2018.8327318.

[10] Paul E. Black, "Ratcliff/Obershelp pattern recognition", in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed. 8 January 2021. (accessed TODAY) Available from: https://www.nist.gov/dads/HTML/ratcliffObershelp.html