UNIVERSITY OF TARTU

Institute of Computer Science

Software Engineering Curriculum

Rene Kütt

Plagiarism Detection Tool for Programming Activity Logs

Master’s Thesis (30 ECTS)

Supervisor(s): Marina Lepp, PhD

Heidi Meier, MSc

Tartu 2023

Plagiarism Detection Tool for Programming Activity Logs

Abstract:

Plagiarism is a critical concern in academia, and educators need effective tools to detect and prevent plagiarism. Currently, most plagiarism detection tools use source code comparison, which is not potent against obfuscation methods used by students. This thesis presents a novel solution for detecting potential plagiarism in programming assignments using logs generated by Thonny IDE and an IntelliJ Platform plugin created as part of this thesis called PALG (Programming Activity Log Generator). A plagiarism detector has been incorporated into Thonny Log Analyser, a web application that processes logs created by Thonny IDE. PALA (Programming Activity Log Analyser) is a web application created by modifying Thonny Log Analyser to be compatible with log files produced by the IntelliJ Platform plugin. The plagiarism detection tool in the web application analyses the logs based on user-specified criteria such as run count, total time spent working, log file size, and pasted text to manually inserted text ratio.

The plagiarism detection tool also includes comparison functionalities that compare log files to each other or log files in different top folders, depending on the chosen analysis type. The comparison includes checking for duplicate files, identical texts pasted in different log files, source codes pasted in different log files and source code similarity detection. The similarity between log files is calculated using Dice's coefficient.

The purpose of the solution is to provide teachers with a quick and efficient overview of potential plagiarism in programming assignments. The solution is designed to work seamlessly with Moodle’s "Download all submissions" action, which produces a ZIP file that can be directly analysed by the web application. The solution's effectiveness is demonstrated through experimental evaluations, and the results indicate its potential to aid teachers in detecting plagiarism in programming assignments efficiently and effectively.

Keywords:

Plagiarism detection, similarity analysis, history-based, log file, web application, Thonny, IntelliJ

CERCS: P175 Informatics, systems theory, S281 Computer-assisted education

Plagiaadikontroll programmeerimise logide põhjal

Lühikokkuvõte:

Plagiaat on akadeemilistes ringkondades kriitiline probleem ja õpetajad vajavad tõhusaid vahendeid plagiaadi tuvastamiseks. Hetkel kasutab enamik plagiaadi kontrollijaid lähtekoodi võrdlust, mis ei ole tõhus õpilaste varjamistehnikate vastu. See lõputöö pakub uudset lahendust võimaliku plagiaadi tuvastamiseks programmeerimisülesannete lahendustes, kasutades sisendiks logisid, mis on genereeritud Thonny IDE poolt või selle lõputöö osana loodud IntelliJ Platform pistikprogrammiga PALG (Programming Activity Log Generator). Lahendus on integreeritud veebirakendusse nimega Thonny Log Analyser, mis võtab sisendlogideks Thonny IDE genereeritud logid. Rakendust Thonny Log Analyser muudeti nii, et see töötaks IntelliJ Platformi pistikprogrammi genereeritud logifailidega ja sellest tulenev veebirakendus kannab nime PALA (Programming Activity Log Analyser). Veebirakenduses olev plagiaadikontroll analüüsib logisid kasutaja määratud kriteeriumide alusel nagu käivitamiste arv, töötamise aeg, logifaili suurus ning kleebitud teksti ja käsitsi sisestatud teksti suhe.

Plagiaadikontroll sisaldab ka võrdlusel põhinevat funktsionaalsust. Rakendus võrdleb sõltuvalt valitud analüüsitüübist logifaile omavahel või erinevates ülemistes kaustades olevaid logifaile. Samuti kontrollib identsete failide olemasolu, identsete tekstide kleepimist, lähtekoodide kleepimist ja lähtekoodide sarnasust. Lähtekoodide sarnasus arvutatakse Dice'i koefitsiendi abil.

Lahenduse eesmärk on anda õpetajatele kiire ja tõhus ülevaade võimalikest plagiaatidest programmeerimistöödes. Lahendus on loodud töötama sujuvalt Moodle'i toiminguga "Salvesta kõik esitatud tööd", mis loob ZIP-faili, mida veebirakendus saab otse analüüsida. Lahenduse tõhusust demonstreeritakse eksperimentaalsete hindamiste kaudu ning tulemused näitavad, et see võib aidata õpetajatel programmeerimisülesannetes plagiaati tõhusalt tuvastada.

Võtmesõnad:

Plagiaadikontroll, sarnasuse analüüs, ajaloopõhine, logifail, veebirakendus, Thonny, IntelliJ

CERCS: P175 Informaatika, süsteemiteooria, S281 Arvuti õpiprogrammide kasutamise metoodika ja pedagoogika

Table of Contents

Introduction

Plagiarism is a serious concern in programming education, where students often submit assignments that may not be entirely their own work. Detecting and preventing plagiarism in programming assignments has become a critical task for educators. Various plagiarism detection solutions have been developed to address this issue, but they often have limitations in terms of accuracy. Students often use obfuscation methods to hide the fact that they have plagiarised, which reduces the efficiency of source code analysis. One way to circumvent some student obfuscations, which hide their plagiarism, is to use a history-based approach to plagiarism detection. History-based algorithms monitor either a user's data from a specific repository or their progress based on activity logs from an IDE. This enables us to monitor students' progress during development and compare it to others.

The Thonny Log Analyser web application is a pre-existing solution that analyses logs generated by the Thonny Integrated Development Environment (IDE) and presents an overview of the development for each log file [1]. Thonny Log Analyser previously enabled teachers to manually go through submitted log files to see activities carried out by students. However, it lacks a built-in plagiarism detection feature, which motivated the development of a new plagiarism detection tool as part of this thesis.

In addition, an IntelliJ Platform plugin called PALG (Programming Activity Log Generator) was created to generate programming activity logs similar to Thonny IDE logs. As part of this thesis, Thonny Log Analyser was altered to work with logs generated by the plugin PALG. The resulting application is named PALA (Programming Activity Log Analyser). The development of the PALG plugin and the PALA application aimed to broaden the solution's capabilities beyond analysing logs exclusively from the Thonny IDE. Specifically, the goal was to create a tool that could generate logs for IntelliJ Platform IDEs and analyse them.

This thesis presents the development and implementation of a plagiarism detection tool that integrates with the Thonny Log Analyser web application. The methodology used for software development, including planning, analysis, design, implementation, and testing, is described in detail. The requirements for the plagiarism detection tool, including functional and non-functional requirements, are also outlined.

The implementation of the plagiarism detection tool and the PALG plugin is discussed in separate sections, highlighting each component's technical aspects and functionalities. Finally, the results obtained from the implementation are presented, as well as the evaluation and validation of the solutions. The thesis also includes a comparison with existing solutions, usability testing with teachers, and possibilities for future work.

Overall, this thesis aims to contribute to the field of plagiarism detection in programming assignments by providing an integrated solution that enhances the capabilities of the Thonny Log Analyser web application. The developed solution is expected to be useful for educators in detecting potential plagiarism in programming assignments, helping to maintain academic integrity and uphold the standards of education.

1 Background

As early as the 1970s, research on source-code plagiarism detection had already been conducted and documented in various papers. However, despite some authors continuing research in the field over the years, it was not until 30 years later that the field gained more popularity [2].

1.1 Plagiarism Detection in Programming Assignments

Plagiarism detection tools significantly lower the time it takes to check for plagiarism in student assignments. At the Georgia Institute of Technology, teachers' time spent checking for plagiarism fell from 50 hours to 10 minutes after adopting the use of MOSS-TAPS, a plagiarism detection tool [3].

A most common way of detecting plagiarism in programming assignments is source-code analysis. Doing this manually for a large group of submissions is almost impossible. There are also many reasons submissions could be very similar, which have nothing to do with plagiarism. For example, using example code provided in the study material, code reuse and small tasks which have little variations in possible solutions [3]. Experiments have shown that most current source-code plagiarism detection tools do not work against obfuscated source-code [4].

There are different algorithms for source-code plagiarism detection, for example, based on style, semantics, history, attribute counting, structure, or string matching. The last three were most used in the literature review [2]. History-based algorithms monitor either a user's data from a specific repository or their progress based on activity logs from an IDE. Some algorithms, like structure-based analysis, have proven to be much more effective than, for example, attribute counting [2]. However, the structure-based analysis also has a higher time complexity.

1.1.1 Obfuscation Methods

In the 2019 systematic literature review of plagiarism detectors by Novak et al., they identified 16 unique obfuscation methods [2]. These were:

1. Visual code formatting
2. Comments modification
3. Translation of program parts

4. Modifying program output
5. Identifier rename
6. Changing constant values
7. Reordering independent lines of code
8. Adding redundant lines of code
9. Splitting up lines of code
10. Merging lines of code
11. Changing of statement specification
12. Replacing control structures with equivalents
13. Simplifying the code
14. Translation of program from other programming language
15. Changing the logic
16. Combining copied and original code

These methods are in the ascending order of completion complexity. Generally, the more complex the obfuscation method is to implement, the more difficult it is to detect. Structural and lexical modifications are the two main categories used to classify obfuscation techniques in the majority of cases. Beginner programmers mainly use lexical changes, as these are easier to implement and do not require an understanding of the code. Advanced programmers are able to use structural changes, which tend to require a deeper understanding of the code and its execution [2].

1.2 Existing Solutions and Their Limitations

In the 2022 literature review of community software by Blanchard et al., they identified 22 open-source plagiarism detection tools, of which 17 are dedicated to code while 7 can be used for plain-text analysis [5]. This text will provide more information about four plagiarism detection tools used in academia. Moodle VPL plagiarism checker was chosen because of Moodle's popularity in academia and how the plagiarism checker is integrated into the system. MOSS-TAPS, ShowYourWork and Vipassana were chosen because of their use of a history-based algorithm which is the same as the plagiarism detection tool created in this thesis.

1.2.1 Moodle VPL Plagiarism Checker

The Virtual Programming Lab (VPL) for Moodle is an open-source tool[1]. The VPL includes tools for plagiarism checking [6]. The procedure for identifying similarities between source files involves three stages: tokenization, comparison, and clustering. Tokenization is a process that generates normalized signatures for each file, facilitating efficient comparison and identification of similarities among them. The process involves three phases: lexical analysis, filtering, and normalization, which extract tokens, remove irrelevant tokens, and standardise expressions into a canonical form to create program signatures. VPL uses three metrics to compare two signatures, producing values between 0.0 to 1.0, where 0.0 means "completely identical" and 1.0 means "completely dissimilar". Using three metrics allows for more effective comparison, as each metric responds differently to code modifications [6].

Moodle VPL plagiarism checker is an excellent solution for detecting similarities. Its only weakness, as for all plagiarism detection tools, which only use source-code comparisons, is that it is vulnerable to obfuscation methods.

1.2.2 MOSS-TAPS

MOSS[2] is a system that automatically determines the similarity of programs, primarily used to detect plagiarism in programming assignments [7]. It accepts batches of documents and returns HTML pages that show where significant sections of a pair of documents are similar. The service currently uses robust winnowing, which is more efficient and scalable than previous algorithms. In addition, MOSS stores positional information with each selected fingerprint and uses these fingerprints to determine where the longest matching sequences are. MOSS can analyse programming languages such as C, C++, Java, C#, Python, JavaScript, and more [7].

The MOSS Tool for Addressing Plagiarism at Scale (MOSS-TAPS)[3] is a plagiarism detection tool that uses MOSS [3]. MOSS-TAPS offers additional features not found in the official MOSS script, including a pure Java implementation for portability, configurable options, preprocessing for nested and ZIP files and submissions in multiple languages, and filtering of results to avoid comparisons of past and current semester projects or current students with themselves. Results are ordered by similarity, with links to colour-coded webpages for each

[1] https://github.com/jcrodriguez-dis/moodle-mod_vpl
[2] https://theory.stanford.edu/~aiken/moss/
[3] https://github.com/danainschool/moss-taps

pair to aid review. Overall, MOSS-TAPS is an effective tool for identifying plagiarism in software assignments [3].

MOSS-TAPS, which is based on a history-based algorithm, may be susceptible to obfuscation techniques since it depends on analysing source code. Nonetheless, its reliance on historical data can make it more challenging for individuals to commit plagiarism without detection.

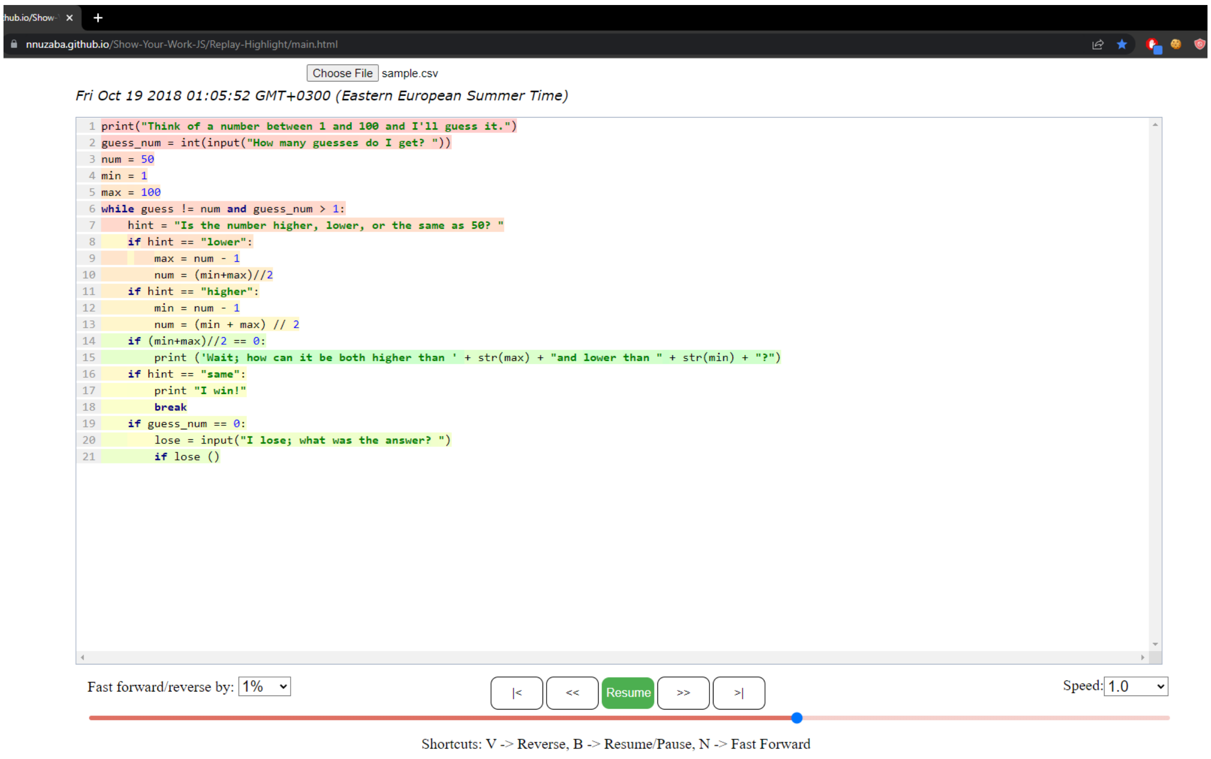
1.2.3 ShowYourWork

ShowYourWork[4] is a plugin for the PyCharm IDE that records user keystrokes to a local file, either in the same directory or a sibling directory (depending on whether the logging group has been submitted or not). The plugin captures two types of data: keystrokes and edits to project files, which can be used to reconstruct the sequence of actions students took while writing their code. Research carried out at Utah State University revealed that gathering keystroke data of students can discourage plagiarism [8]. Most students reacted positively, believing it created a fairer environment for everyone. However, the study also indicated that honest students might experience heightened anxiety levels if they are not adequately informed about the functioning of plagiarism detection techniques [8].

ShowYourWork plugin generated logs can be replayed by using the Show Your Work JS program[5] (Figure 1). It allows the user to replay the actions made by the student in the IDE [9]. Finding plagiarisms by using Show Your Work JS still requires a lot of effort as teachers have to manually replay all files, and at most, it is possible to find large chunks of code which were pasted.

[4] https://plugins.jetbrains.com/plugin/18353-showyourwork

[5] https://github.com/nnuzaba/Show-Your-Work-JS/tree/master/Replay-Highlight


nnuzaba.github.io/Show-Your-Work-JS/Replay-Highlight/main.html
Choose File
sample.csv
Resume
|<
<<
>>
>|

Fri Oct 19 2018 01:05:52 GMT+0300 (Eastern European Summer Time)

```
print("Think of a number between 1 and 100 and I'll guess it.")
guess_num = int(input("How many guesses do I get? "))
num = 50
min = 1
max = 100
while guess != num and guess_num > 1:
    hint = "Is the number higher, lower, or the same as 50? "
    if hint == "lower":
        max = num - 1
        num = (min+max)//2
    if hint == "higher":
        min = num - 1
        num = (min + max) // 2
    if (min+max)//2 == 0:
        print ('Wait; how can it be both higher than ' + str(max) + "and lower than " + str(min) + "?")
    if hint == "same":
        print "I win!"
        break
    if guess_num == 0:
        lose = input("I lose; what was the answer? ")
        if lose ()
```

Fast forward/reverse by: 1%

Speed: 1.0

Shortcuts: V -> Reverse, B -> Resume/Pause, N -> Fast Forward

Figure 1. Snapshot of Show Your Work JS Replay-Highlight functionality.

1.2.4 Vipassana

Vipassana is a history-based plagiarism detection tool [10]. Vipassana is a tool developed at the Florida Institute of Technology that allows instructors to distribute and collect assignments via online repositories. Students can access the projects, download the required files, and upload their work as they progress. Vipassana records each event, including all files containing source code, and uses MOSS to determine the originality of each submission. In addition, instructors can view student activity and progress through the Vipassana Web tool, including a bar chart representing each commit and a graph showing code growth over time. Vipassana is effective in deterring plagiarism and allows instructors to quickly identify which students have started early and which ones submit their work at the last minute [10].

Vipassana is vulnerable to obfuscation methods as it relies on source-code analysis. However, this history-based plagiarism detection software does make it more difficult to plagiarise with ease.

2 Pre-existing Functionalities in Thonny Log Analyser

Thonny Log Analyser was created by the author for his Bachelor's thesis [1]. Afterwards, additional improvements were made to the project through a contractual agreement with the university. Thonny Log Analyser uses the logs generated by the Thonny IDE to output analysis results for every log file. The analyser outputs general information about log files like start time, end time, elapsed time, run count, error count, pasted text count, debug count, files created, files ran, and files opened, which gives an excellent broad overview of the work done. The analyser also allows us to see pasted texts, error messages and a program length graph, which shows the timeline change of text length in the IDE. For example, the program length graph could be used to see the text change for each program opened in the log file and the total length and shell text length changes. The analyser also provides a replayer which can be used to get a more fine-grained overview of what changes were made in the IDE. The replayer allows the user to move through log file actions and see how the state of the IDE changed in time. The replayer also has an auto-play feature where the replayer automatically iterates through the log actions and updates the IDE at the user-specified speed. The analyser also provides a download analysis results feature, where a CSV file is generated for all log files. For every log file, a row is generated with all general info values, including counts for several different error types, pasted change lengths and several other metrics.

One of Thonny Log Analyser's most powerful features is the ability recursively open the submitted ZIP file(s). Then, analyse the logs and group them by top-level folders if the grouped output is chosen. This feature integrates well with Moodle's "Download all submissions" action, which produces a ZIP file that the Thonny Log Analyser can directly analyse. Furthermore, since the ZIP file's top-level folders are named after the students, the grouped output will group analysed log files under the respective students' names.

Before the plagiarism detection tool was added to Thonny Log Analyser, indications of plagiarism could be found manually iterating through analysis results. Some indicators of plagiarism include incomplete log files, for example, a log file that is too small, elapsed time is too small or run count was low, all indicating that development had not taken place in the scope of the log file. Furthermore, signs of plagiarism could be uncovered by analysing the pasted texts and seeing discrepancies or using the replayer to spot abnormal activities. Also, the replayer can be used to look at the source code produced and manually find similarities to other students.

3 Methodology

This chapter discusses the software development life cycle used to create the plagiarism detection tool for Thonny Log Analyser and the IntelliJ Platform plugin PALG (Programming Activity Log Generator). Furthermore, this chapter will provide the functional and non-functional requirements for the applications mentioned above and the web application PALA (Programming Activity Log Analyser).

3.1 Plagiarism Detection Tool

The Agile Software Development Life Cycle was chosen for plagiarism detection development as it best facilitated the changing needs and requirements of the project. Initial requirements and goals were agreed upon with the thesis supervisors. After starting the implementation, it became more apparent what could be accomplished with reasonable execution time complexity, after which requirements were refined. A Minimum Viable Product (MVP) was developed and deployed to get supervisor feedback. After receiving feedback for the MVP, the plagiarism detection tool was incrementally refined, developed, and deployed in subsequent iterations. After each iteration, the thesis supervisors manually tested the tool to ensure it met the desired quality standards and fulfilled the requirements. This iterative development approach allowed the plagiarism detection tool to evolve, based on supervisor feedback, and to better address the changing needs and requirements of the project.

3.1.1 Functional Requirements

The following functional requirements were defined and refined for the plagiarism detection tool:

Req.1: The plagiarism detection tool shall be integrated into the Thonny Log Analyser web application.

Req.2: The solution shall compare log files to each other or to all logs in top folders based on the type of analysis selected.

Req.3: The solution shall analyse each log file for the run count, total time spent working, log file size, and pasted text to manually inserted text ratio.

Req.4: The solution shall compare log files for duplicate submissions, identical pasted texts, identical source code pasted in different log files, and source code similarity.

Req.5: The solution shall allow the user to specify the minimum value for each analysed metric for plagiarism detection.

Req.6: The solution shall provide a quick overview for teachers to determine whether plagiarism might have occurred.

3.1.2 Non-functional Requirements

The following non-functional requirements were defined and refined for the plagiarism detection tool:

NFReq.1: The solution shall have a user-friendly interface that is easy to navigate.

NFReq.2: The solution shall provide clear and concise results for teachers to easily interpret.

NFReq.3: The solution shall be stable and reliable, avoiding any unexpected crashes or errors.

NFReq.4: The solution shall analyse a ZIP file with a size of 10MB for plagiarism in less than 30 seconds.

NFReq.5: The solution shall be compatible with Google Chrome, Microsoft Edge and Firefox browsers.

3.2 PALG

Since the requirements for PALG were clear and well-defined, the Waterfall model was a suitable development methodology for the project. The main goal was to create a plugin that would mimic the logging process of the existing Thonny IDE, and this objective remained stable throughout the project.

Due to the short duration of the project and the clarity of the requirements, intermittent testing and supervisor feedback were not required, and development could progress more quickly. However, before the plugin was released, it was validated with the supervisors to ensure it met their expectations and was functioning correctly.

The author manually tested the plugin during the development process to ensure it was functioning as intended and met the project's requirements. This testing was done in conjunction with the development process to ensure that any issues or bugs were caught and addressed promptly and to ensure that the final product met the required quality standards.

3.2.1 Functional Requirements

The following functional requirements were defined and refined for the IntelliJ Platform plugin PALG:

Req.1: The plugin should be compatible with the latest version of IntelliJ Platform IDEs.

Req.2: The plugin should create a new JSON log every time an IntelliJ Platform project is opened.

Req.3: The plugin should generate JSON log files in a compatible format with Thonny IDE’s generated logs.

Req.4: The plugin should log user actions such as text typed, text pasted, text deleted, file ran, file debugged, file switched, file opened, and file closed.

Req.5: The plugin should provide a button to navigate to the directory where JSON files are generated.

3.2.2 Non-functional Requirements

The following non-functional requirements were defined and refined for the IntelliJ Platform plugin PALG:

NFReq.1: The plugin should have a minimal impact on the performance of IntelliJ Platform IDEs.

NFReq.2: The plugin should follow the best practices for IntelliJ IDEA plugin development.

NFReq.3: The plugin should be easy to install and configure.

3.3 Requirements for the PALA Application

The PALA application must conform to all the functional and non-functional requirements established for the plagiarism detection tool. This chapter additionally outlines the specific functional requirements that the PALA application must satisfy to effectively meet the needs of its users.

3.3.1 Functional Requirements

The following functional requirements were defined and refined for the PALA application:

Req.1: PALA shall be able to analyse files in the format of logs created by the PALG IntelliJ plugin.

Req.2: PALA shall provide all the same functionalities that Thonny Log Analyser provides including the added plagiarism detection tool.

4 Implementation of the Plagiarism Detection Tool

This chapter describes the implementation details of the plagiarism detection tool for Thonny Log Analyser. The tool was written using JavaScript, HTML, and CSS, along with jQuery, JSZip[6] , crc-32[7], and string-similarity[8] libraries and the Bootstrap framework.

The plagiarism detection tool employs the crc-32 (Cyclic Redundancy Check) algorithm to calculate the checksums of files, enabling it to detect duplicate files. The JSZip library calculates the checksums of log files within ZIP files, while the crc-32 library calculates the checksums of log files that are not within ZIP files. If a match is detected during the comparison of checksums, it indicates that the two files are identical.

If the log files are analysed as grouped output, meaning that log files are grouped by top-level folders, then the plagiarism detection tool compares logs with logs in different top-level folders. If the files are not analysed as grouped output, all log files are compared.

The plagiarism detection tool analyses student work based on the run count, time spent working, and log file size. If the files are analysed as grouped output, then student work metrics are summed, which are in the same top-level folder. Otherwise, student work is analysed for each log file. Finally, they are analysed against user-specified metrics.

The plagiarism detection tool finds the percentage of pasted texts to typed texts. It finds the ratio of pasted texts to typed texts for each source code file contained in log files and outputs the ones higher than the user-specified metrics.

The plagiarism detection tool finds pasted texts identical to source codes in other students' source code files. Pasted texts are added to a map based on their length, where they are compared with already added pasted texts with the same length, and matches are grouped. The source codes from log files are added to pasted text's map if there is a match from the pasted texts. Afterwards, the map is filtered to remove pasted texts with no matches to source codes.

The plagiarism detection tool compares log files' source codes using a library called string-similarity. String-similarity works by finding the degree of similarity between two strings based on Dice's coefficient. The *compareTwoStrings* function of the string-similarity library returns a fraction between 0 and 1, indicating the similarity between the two strings. 0 indicates

[6] https://www.jsdelivr.com/package/npm/jszip
[7] https://www.jsdelivr.com/package/npm/crc-32
[8] https://www.jsdelivr.com/package/npm/string-similarity

completely different strings, and 1 indicates identical strings. Similar logs will be displayed when the returned value is higher or equal to the user-specified similarity percentage. The comparison is case-sensitive. Log files may contain multiple source codes, depending on how many files were worked on during the log file. When students submit log files, they may also submit those that might not be relevant. Therefore, in a class of 30 students, if everyone submits multiple log files containing multiple source code files, the total sum of source codes can easily be over a thousand. The *compareTwoStrings* function has a time complexity of O(n**2), where n is the length of strings being compared. Due to this, comparing all source codes would be excessively time-consuming. Instead, source codes are sorted by length, and only the y closest (from both sides of the selected source code) are compared, where y is calculated with the formula:

$$y = \{ 5 \quad if\ 10000/x < 5, \quad 10000/x\ \ if\ 10000/x >= 5 \}$$

where x is the length of the source code array. The result of the formula is inversely proportional to the size of the array. This formula ensures that the number of comparisons remains reasonable even for large numbers of log files and source codes. Once the source codes are sorted, and the subarrays are selected for comparison, the *compareTwoStrings* function is used to find the degree of similarity between the source codes.

The user specified plagiarism metrics, and their defaults are the following:

- Pasted text minimum length: 50 characters (default)
- Pasted to typed texts percentage: 80% (default)
- Source code minimum length: 100 characters (default)
- Source code similarity percentage: 90% (default)
- Run count is equal or less than: 0 executions (default)
- Total time working is less than (minutes): 15 minutes (default)
- Total log file size less than (KB): 100 KB (default)

5 Implementation of the IntelliJ Platform Plugin PALG

This chapter discusses the implementation of the Programming Activity Log Generator (PALG), an IntelliJ Platform Plugin that logs user actions into JSON log files. The purpose of this plugin is to track the user's activity within the IDE in a compatible format with Thonny Log Analyser.

The JSON log files are saved into the java.io.tmpdir subfolder named PALG. The system property java.io.tmpdir is used to specify the default temporary directory that should be used by the Java Virtual Machine (JVM) when creating temporary files. This directory is platform-dependent, and its location varies depending on the operating system being used. The plugin logs the following user actions when the corresponding listener is called:

- Text inserted: When text is typed or pasted into the editor, the plugin logs the action along with the text content.
- File ran: When a program is executed, the plugin logs the action and the program name.
- File debugged: When a program is debugged, the plugin logs the action and the program name.
- File switched: When a user switches between files, the plugin logs the action along with the file name.
- File opened: When a file is opened, the plugin logs the action along with the file name.
- File closed: When a file is closed, the plugin logs the action and the file name.

The PALG plugin is a Gradle Kotlin based IntelliJ Platform plugin. The plugin uses the logback[9] and kotlin-logging[10] frameworks for logging actions. The gson library is used to serialize the activity data object to JSON. The plugin also uses several listener classes to capture user actions:

- FileEditorManagerListener: This class listens for file opening, closing, and switching.
- DocumentListener: This class listens for changes in the document, such as text insertion or deletion.
- CopyPastePreProcessor: This class listens for text pasting events.
- ExecutionListener: This class listens for program execution and debugging events.

[9] https://logback.qos.ch/
[10] https://github.com/oshai/kotlin-logging

After completing the PALG plugin, the Thonny Log Analyser was forked and updated to be compatible with the logs generated by the PALG plugin.

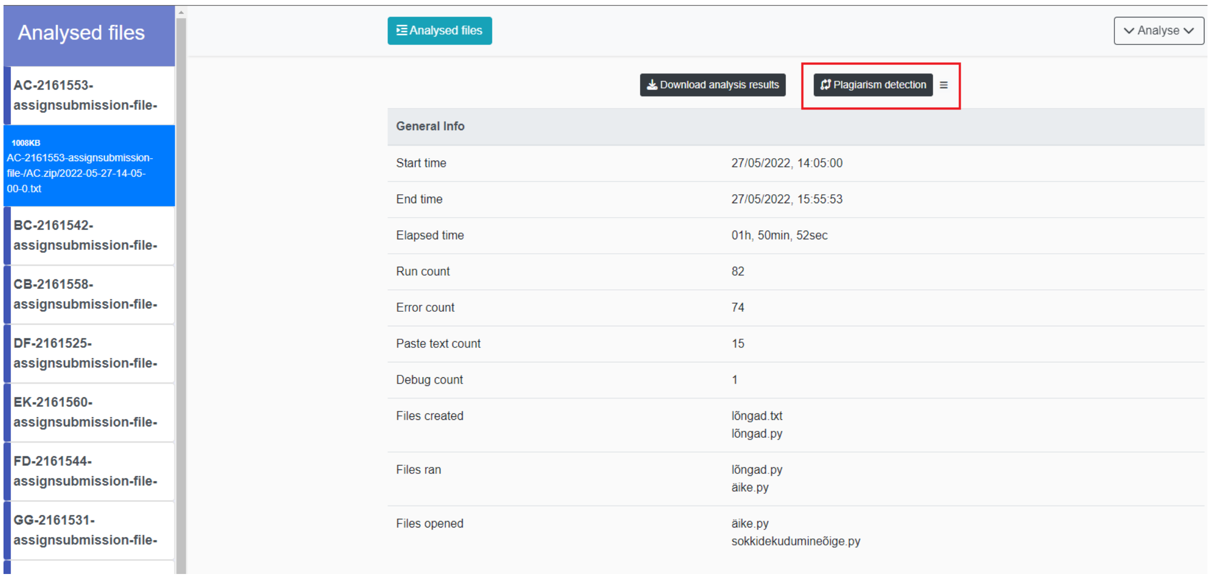
In conclusion, the Programming Activity Log Generator (PALG) plugin provides a simple yet effective solution for tracking and analysing user activity within the IntelliJ Platform IDEs. By implementing listener classes and using the gson library for serialization, the plugin is able to capture a wide range of user actions and generate JSON log files in a compatible format with Thonny Log Analyser.

6 Results

This chapter presents the results of the created plagiarism detection tool, programming activity log generator (PALG) and programming activity log analyser (PALA).

6.1 Plagiarism Detection Tool

The created plagiarism detection tool can be accessed through the following link: https://progtugi.cs.ut.ee/thonny-log-analyser/. The version control system used throughout the development process was Bitbucket[11]. After analysing log files, the main screen displays a plagiarism detection button and a toggle button for specifying the metrics (Figure 2).



Analysed files

AC-2161553-assignsubmission-file-

1008KB
AC-2161553-assignsubmission-file-/AC.zip/2022-05-27-14-05-00-0.txt

BC-2161542-assignsubmission-file-

CB-2161558-assignsubmission-file-

DF-2161525-assignsubmission-file-

EK-2161560-assignsubmission-file-

FD-2161544-assignsubmission-file-

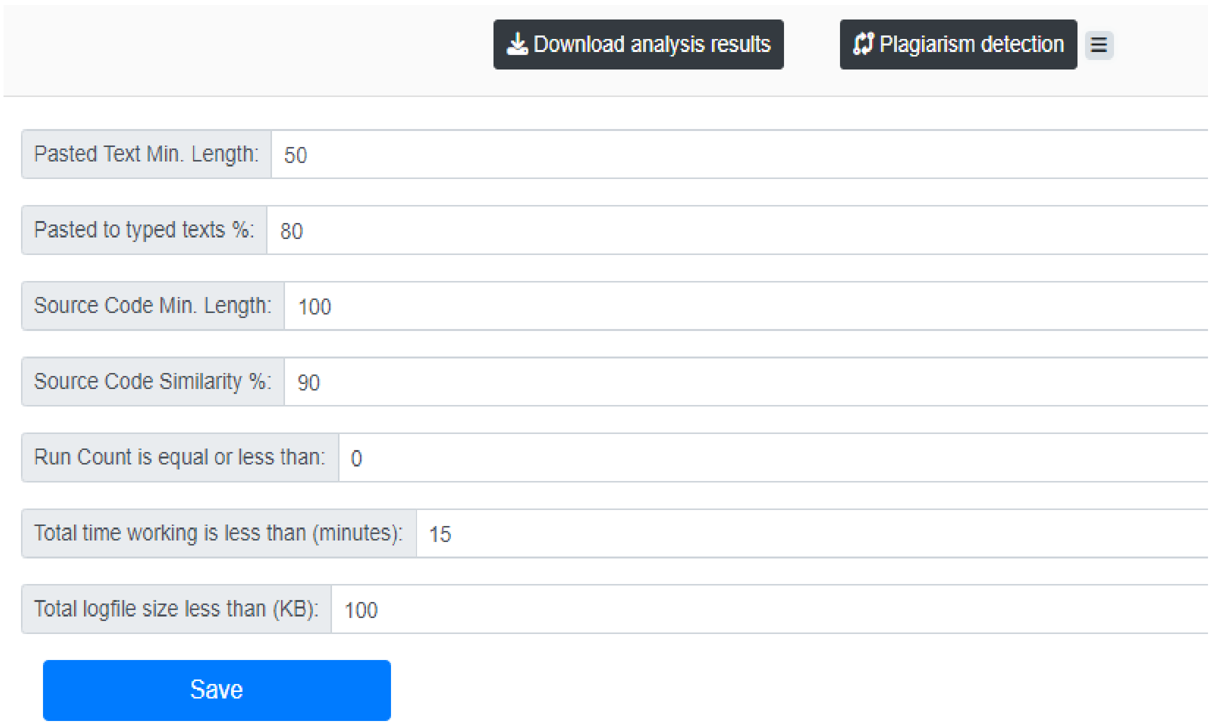
GG-2161531-assignsubmission-file-

General Info	
Start time	27/05/2022, 14:05:00
End time	27/05/2022, 15:55:53
Elapsed time	01h, 50min, 52sec
Run count	82
Error count	74
Paste text count	15
Debug count	1
Files created	lõngad.txt lõngad.py
Files ran	lõngad.py äike.py
Files opened	äike.py sokkidekudumineõige.py

Figure 2. Thonny Log Analyser main screen displaying analysed log files.

Plagiarism detection metrics can be edited after clicking the toggle button to specify the metrics (Figure 3).

[11] https://bitbucket.org/renekutt/thonny-log-analyser



Pasted Text Min. Length:	50
Pasted to typed texts %:	80
Source Code Min. Length:	100
Source Code Similarity %:	90
Run Count is equal or less than:	0
Total time working is less than (minutes):	15
Total logfile size less than (KB):	100

Figure 3. Plagiarism detection metrics form.

The metrics form will be saved after clicking the save button. Initially, the form is populated by the default metric values. The log files are analysed for plagiarism according to the user-specified metrics by clicking on the plagiarism detection button. After which, the plagiarism detection tool will open in an overlay modal (Figure 4).

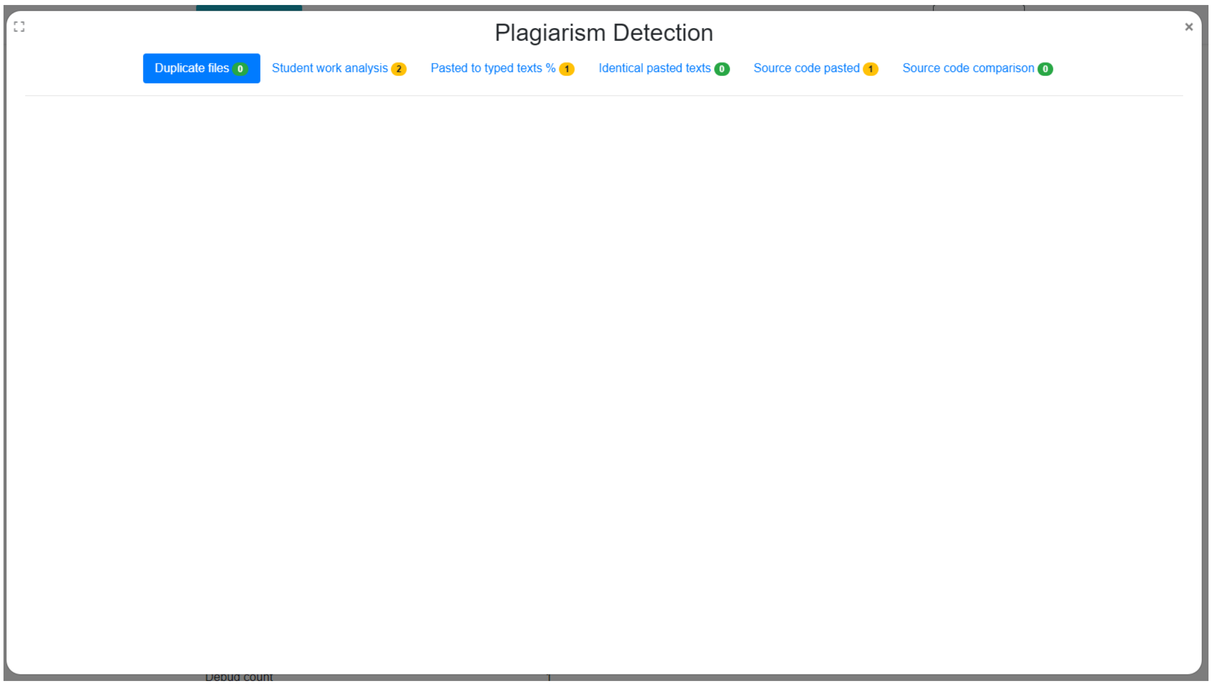


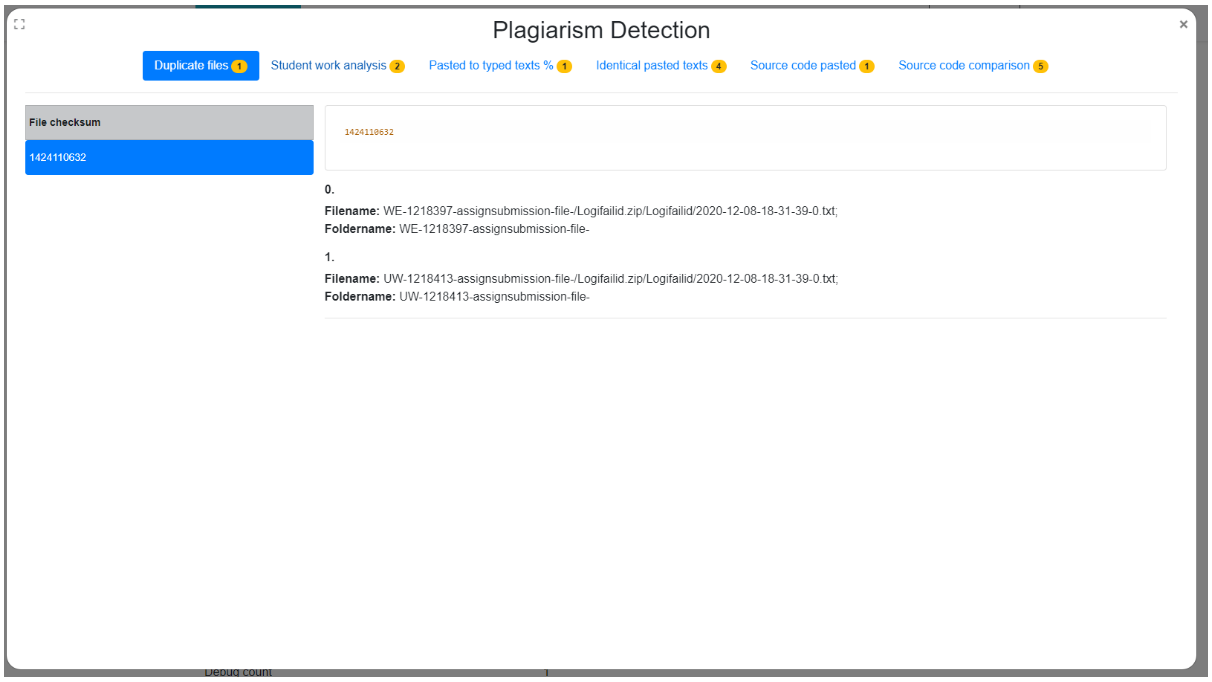
Figure 4. Plagiarism detection tool.

In the opened plagiarism detection tool modal, the navigation bar will display all the analysis types with numbers next to them indicating how many findings are under the given plagiarism analysis type. The number will be displayed as green if the number of findings is 0. Otherwise, it will be coloured yellow. Analysis types can be navigated by clicking on the navigation analysis type wished to be open. The duplicate files tab is set to active by default on the opening of the plagiarism detection tool. Plagiarism analysis results will be displayed under the navigation bar.

The results window has a set structure across all analysis types. All the findings are displayed in a list on the left sidebar (Figure 5). The sidebar header explains the types of values contained in the sidebar. Sidebar findings can be navigated by clicking on list items. Alternatively, the tab key can be used to move to the following list item or the shift + tab key to move to the previous list item; upon activating a sidebar list item the main results window updates accordingly.

6.1.1 Duplicate Files Analysis

The duplicate files analysis finds all the duplicate files by comparing the files checksums which indicate the files are identical. The analysis results display all the duplicate file checksums in the sidebar (Figure 5).



```
1424110632
```

0.

Filename: WE-1218397-assignsubmission-file-/Logifailid.zip/Logifailid/2020-12-08-18-31-39-0.txt;
Foldername: WE-1218397-assignsubmission-file-

1.

Filename: UW-1218413-assignsubmission-file-/Logifailid.zip/Logifailid/2020-12-08-18-31-39-0.txt;
Foldername: UW-1218413-assignsubmission-file-

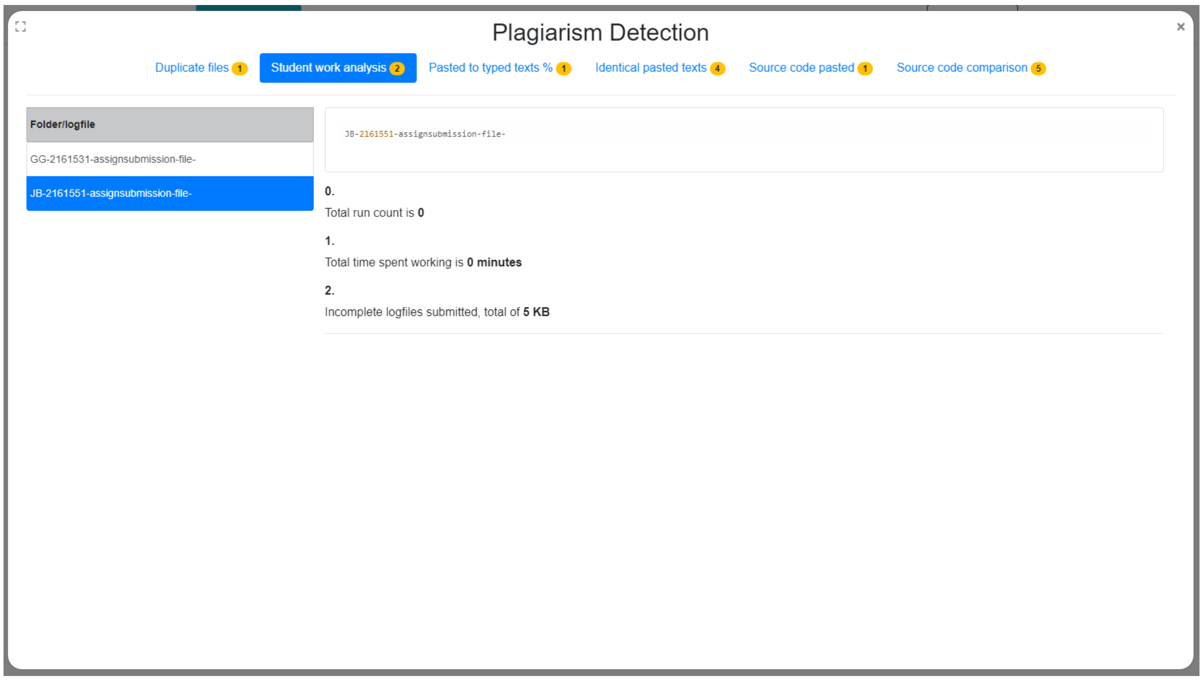
Figure 5. Plagiarism detection duplicate files analysis.

In the duplicate files results window, the checksum value is shown at the top, while the matching log file values are shown below. Log file values display the file name and also the folder name if grouped output analysis was chosen.

Duplicate files analysis provides near definitive proof that plagiarism has occurred, as the likelihood that two different log files have the same checksum is extremely low. That means log files were shared between students.

6.1.2 Student Work Analysis

Student work analysis finds all log files where run count, time spent working or log file size is smaller than the default or user-specified metrics (Figure 6). If the log files are analysed with grouped output, then the log file metrics are also grouped according to top-level folders. That means the total amount of work for a given student is checked against the metrics.



GG-2161531-assignsubmission-file-
JB-2161551-assignsubmission-file-

```
JB-2161551-assignsubmission-file-
```

0.
Total run count is **0**

1.
Total time spent working is **0 minutes**

2.
Incomplete logfiles submitted, total of **5 KB**

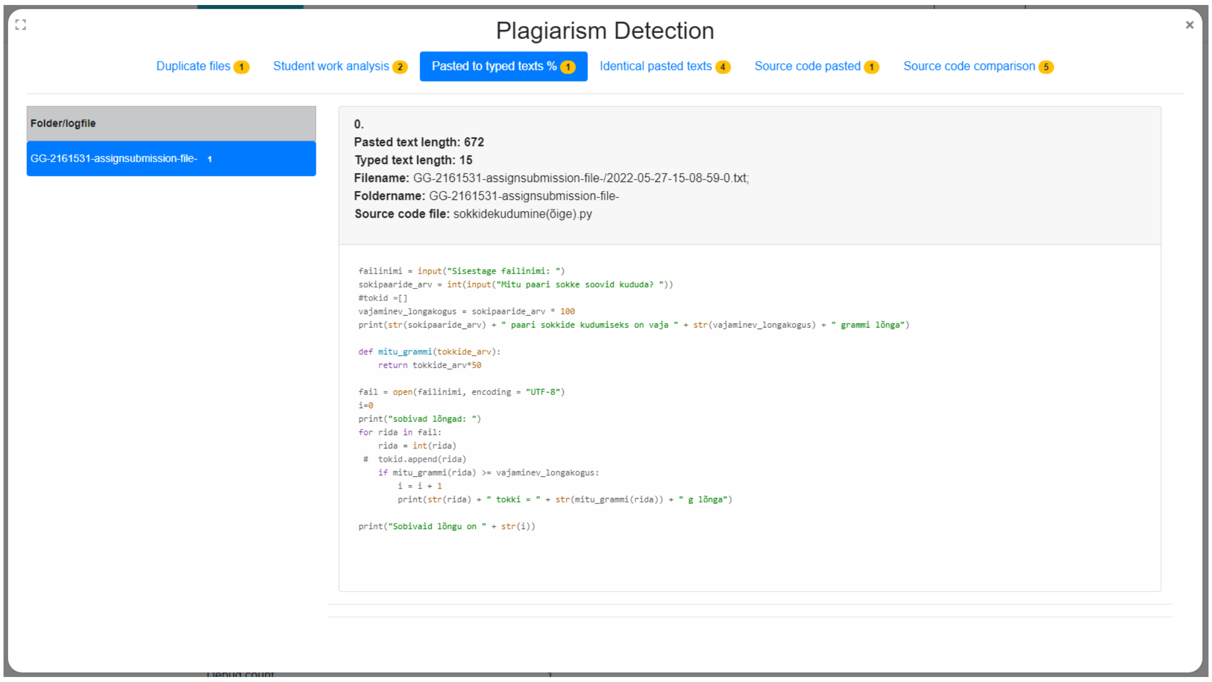
Figure 6. Plagiarism detection student work analysis.

The findings of the student work analysis are presented in the sidebar, where the list item values are either log file names or folder names, depending on if the grouped output was chosen. In the results window, the log file or folder name is shown at the top, while the findings are shown below. The findings display all of the metrics lower than the system-specified metrics in an ordered list.

Student work analysis provides an overview of incomplete log files. For example, a low run count indicates that the student did not test the solution. Additionally, low time spent working may warrant attention as it could potentially indicate plagiarism or that the student submitted the wrong log files. Finally, a small log file size indicates that proper development has not taken place while the logger has been active.

6.1.3 Pasted to Typed Texts Percentage

The pasted to typed texts analysis finds all source code files inside the log files where the ratio of pasted texts to typed texts is bigger than the default or user-specified metric (Figure 7).



0.
Pasted text length: 672
Typed text length: 15
Filename: GG-2161531-assignsubmission-file-/2022-05-27-15-08-59-0.txt;
Foldername: GG-2161531-assignsubmission-file-
Source code file: sokkidekudumine(õige).py

```
failinimi = input("Sisestage failinimi: ")
sokipaaride_arv = int(input("Mitu paari sokke soovid kududa? "))
#tokid =[]
vajaminev_longakogus = sokipaaride_arv * 100
print(str(sokipaaride_arv) + " paari sokkide kudumiseks on vaja " + str(vajaminev_longakogus) + " grammi lõnga")

def mitu_grammi(tokkide_arv):
    return tokkide_arv*50

fail = open(failinimi, encoding = "UTF-8")
i=0
print("sobivad lõngad: ")
for rida in fail:
    rida = int(rida)
 #  tokid.append(rida)
    if mitu_grammi(rida) >= vajaminev_longakogus:
        i = i + 1
        print(str(rida) + " tokki = " + str(mitu_grammi(rida)) + " g lõnga")

print("Sobivaid lõngu on " + str(i))
```

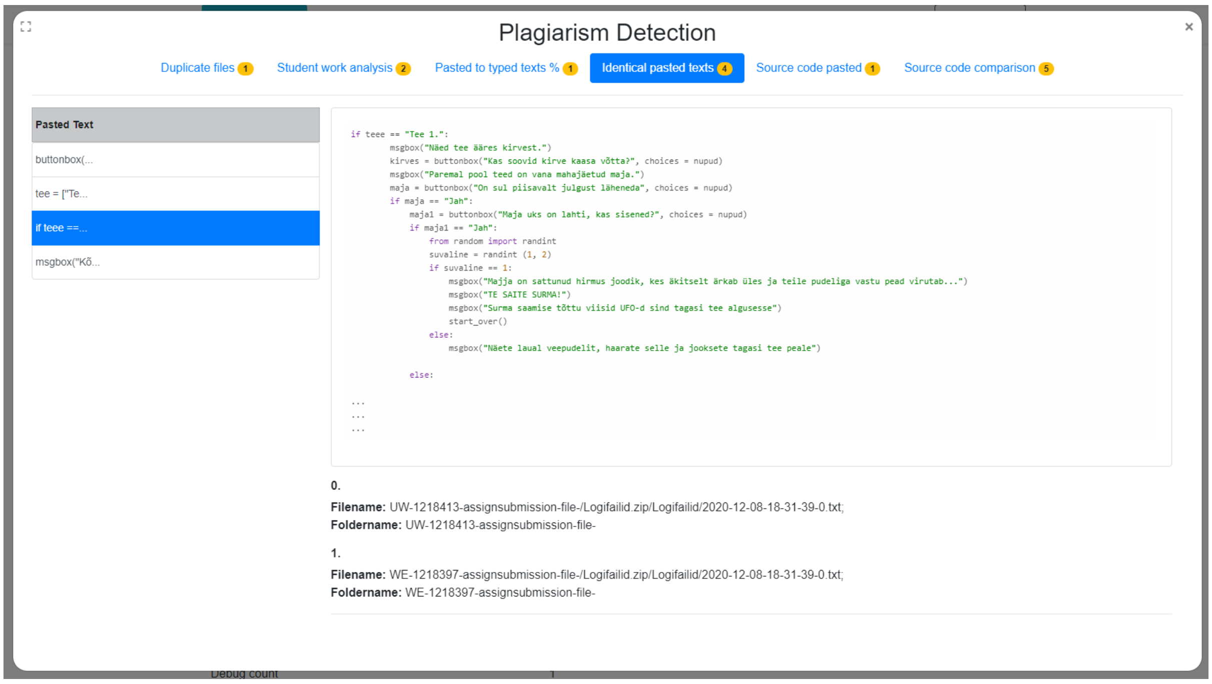
Figure 7. Plagiarism detection pasted to typed texts percentage.

If files were analysed using grouped output, the log file findings would be grouped under top-level folders. Otherwise, all log file findings will be separate. The values in the sidebar are either folder names or log file names, depending on the file grouping. Each source code file finding is organised using Bootstrap card components in the results window. The card header contains analysis results, and the card body contains the source code file's contents. The analysis results in the header include pasted text length, typed text length, file name, folder name (if the grouped output was chosen) and source code file name.

The pasted to typed texts analysis results help identify source code files where most of the work was done by pasting. This does not mean that the student has plagiarised, but it indicates to the teacher that the log file should be looked over in more detail.

6.1.4 Identical Pasted Texts

The identical pasted texts analysis finds all pasted texts in different log files which are identical (Figure 8). If grouped output is chosen, log file pasted texts are compared with pasted texts from different top-level folder log files.



```
if teee == "Tee 1.":
        msgbox("Näed tee ääres kirvest.")
        kirves = buttonbox("Kas soovid kirve kaasa võtta?", choices = nupud)
        msgbox("Paremal pool teed on vana mahajäetud maja.")
        maja = buttonbox("On sul piisavalt julgust läheneda", choices = nupud)
        if maja == "Jah":
            maja1 = buttonbox("Maja uks on lahti, kas sisened?", choices = nupud)
            if maja1 == "Jah":
                from random import randint
                suvaline = randint (1, 2)
                if suvaline == 1:
                    msgbox("Majja on sattunud hirmus joodik, kes äkitselt ärkab üles ja teile pudeliga vastu pead virutab...")
                    msgbox("TE SAITE SURMA!")
                    msgbox("Surma saamise tõttu viisid UFO-d sind tagasi tee algusesse")
                    start_over()
                else:
                    msgbox("Näete laual veepudelit, haarate selle ja jooksete tagasi tee peale")

            else:

...
...
...
```

0.

Filename: UW-1218413-assignsubmission-file-/Logifailid.zip/Logifailid/2020-12-08-18-31-39-0.txt;
Foldername: UW-1218413-assignsubmission-file-

1.

Filename: WE-1218397-assignsubmission-file-/Logifailid.zip/Logifailid/2020-12-08-18-31-39-0.txt;
Foldername: WE-1218397-assignsubmission-file-

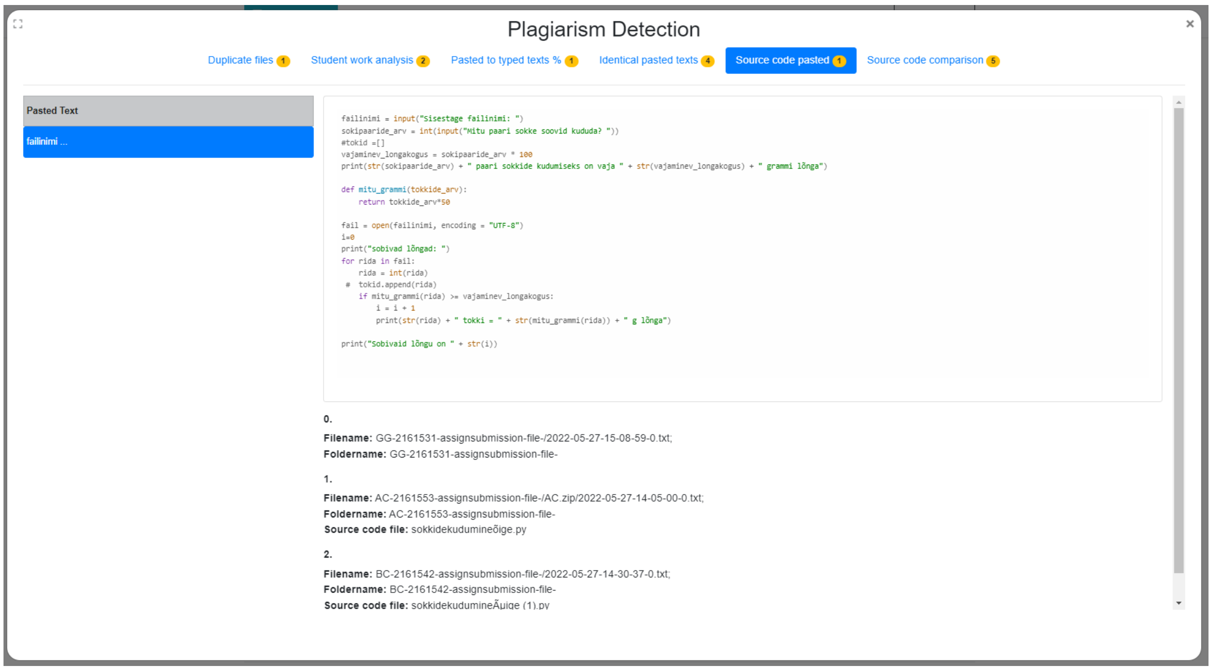
Figure 8. Plagiarism detection identical pasted texts.

The findings of identical pasted texts are presented in the sidebar, where list item values are the first ten characters of the pasted text. In the result window, the full pasted text is displayed at the top, while the log files' details are shown below. Log file values display the file name and also the folder name if grouped output analysis was chosen.

The identical pasted texts analysis results help identify instances where the same code has been pasted. While the identical pasted texts do not necessarily confirm plagiarism, it does signal the teacher to review the corresponding log file carefully.

6.1.5 Source Code Pasted

The source code pasted analysis finds all pasted texts which are identical to the contents of other log files' source codes (Figure 9). If grouped output is chosen, the source codes will not be matched to any pasted texts found in the log files of the given top-level folder.



```
failinimi = input("Sisestage failinimi: ")
sokipaaride_arv = int(input("Mitu paari sokke soovid kududa? "))
#tokid =[]
vajaminev_longakogus = sokipaaride_arv * 100
print(str(sokipaaride_arv) + " paari sokkide kudumiseks on vaja " + str(vajaminev_longakogus) + " grammi lõnga")

def mitu_grammi(tokkide_arv):
    return tokkide_arv*50

fail = open(failinimi, encoding = "UTF-8")
i=0
print("sobivad lõngad: ")
for rida in fail:
    rida = int(rida)
 #  tokid.append(rida)
    if mitu_grammi(rida) >= vajaminev_longakogus:
        i = i + 1
        print(str(rida) + " tokki = " + str(mitu_grammi(rida)) + " g lõnga")

print("Sobivaid lõngu on " + str(i))
```

0.

Filename: GG-2161531-assignsubmission-file-/2022-05-27-15-08-59-0.txt;
Foldername: GG-2161531-assignsubmission-file-

1.

Filename: AC-2161553-assignsubmission-file-/AC.zip/2022-05-27-14-05-00-0.txt;
Foldername: AC-2161553-assignsubmission-file-
Source code file: sokkidekudumineõige.py

2.

Filename: BC-2161542-assignsubmission-file-/2022-05-27-14-30-37-0.txt;
Foldername: BC-2161542-assignsubmission-file-
Source code file: sokkidekudumineÃµige (1).py

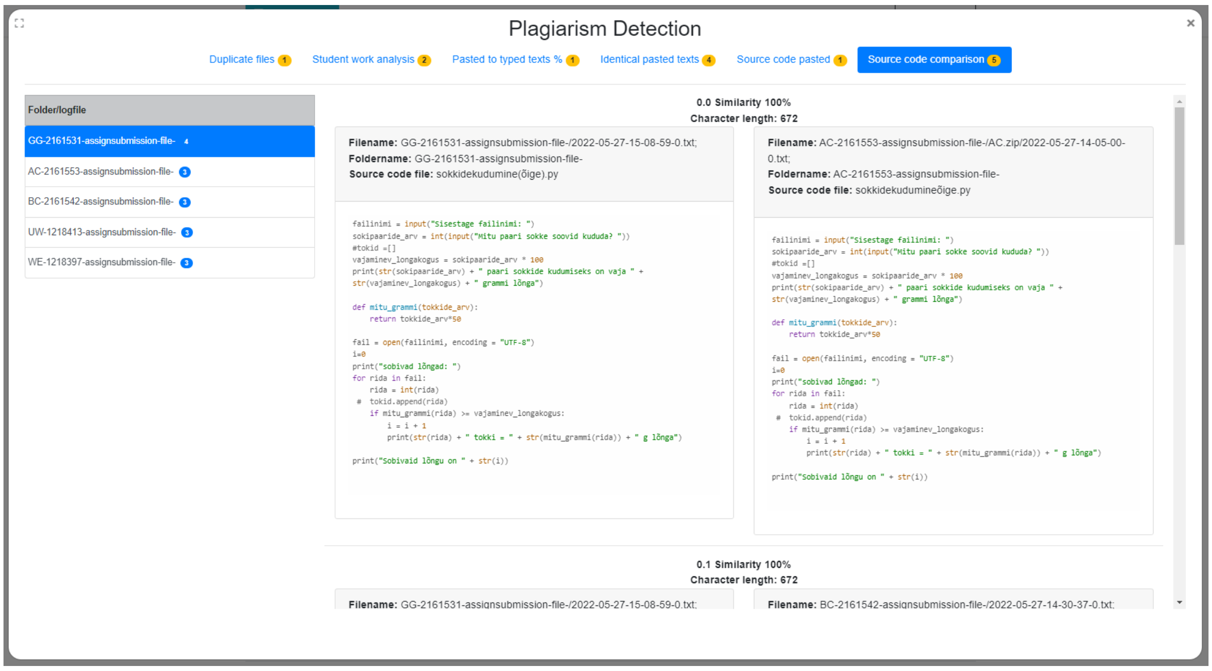
Figure 9. Plagiarism detection source code pasted.

The findings of the source code pasted analysis are presented in the sidebar, where list item values are the first ten characters of the pasted text. The result window displays the complete pasted text at the top and the log file details below, showing the file name and the folder name if the grouped output analysis was selected. Additionally, the source code file name will be displayed for the log files where the matching source code was located. Otherwise, if the source code file name attribute is absent, it means that the identical text was pasted in the given log file.

The source code pasted analysis can help detect cases of plagiarism where students may have copied code from other students' log files. This can aid teachers in identifying students who may need further attention and monitoring.

6.1.6 Source Code Comparison

The source code comparison finds source codes whose Dice's coefficient is over the specified similarity percentage (Figure 10). The source codes are sorted by length into an array, and the closest y from both sides of the selected source code are compared. The variable y is inversely proportional to the size of the array to keep the time complexity of the process low. If grouped output is chosen, source codes are compared with source codes from different top-level folder log files.


Plagiarism Detection
Duplicate files 1
Student work analysis 2
Pasted to typed texts % 1
Identical pasted texts 4
Source code pasted 1
Source code comparison 5
Folder/logfile
GG-2161531-assignsubmission-file- 4
AC-2161553-assignsubmission-file- 3
BC-2161542-assignsubmission-file- 3
UW-1218413-assignsubmission-file- 3
WE-1218397-assignsubmission-file- 3

0.0 Similarity 100%
Character length: 672

Filename: GG-2161531-assignsubmission-file-/2022-05-27-15-08-59-0.txt;
Foldername: GG-2161531-assignsubmission-file-
Source code file: sokkidekudumine(õige).py

```
failinimi = input("Sisestage failinimi: ")
sokipaaride_arv = int(input("Mitu paari sokke soovid kududa? "))
#tokid =[]
vajaminev_longakogus = sokipaaride_arv * 100
print(str(sokipaaride_arv) + " paari sokkide kudumiseks on vaja " +
str(vajaminev_longakogus) + " grammi lõnga")

def mitu_grammi(tokkide_arv):
    return tokkide_arv*50

fail = open(failinimi, encoding = "UTF-8")
i=0
print("sobivad lõngad: ")
for rida in fail:
    rida = int(rida)
 #  tokid.append(rida)
    if mitu_grammi(rida) >= vajaminev_longakogus:
        i = i + 1
        print(str(rida) + " tokki = " + str(mitu_grammi(rida)) + " g lõnga")

print("Sobivaid lõngu on " + str(i))
```

Filename: AC-2161553-assignsubmission-file-/AC.zip/2022-05-27-14-05-00-0.txt;
Foldername: AC-2161553-assignsubmission-file-
Source code file: sokkidekudumineõige.py

```
failinimi = input("Sisestage failinimi: ")
sokipaaride_arv = int(input("Mitu paari sokke soovid kududa? "))
#tokid =[]
vajaminev_longakogus = sokipaaride_arv * 100
print(str(sokipaaride_arv) + " paari sokkide kudumiseks on vaja " +
str(vajaminev_longakogus) + " grammi lõnga")

def mitu_grammi(tokkide_arv):
    return tokkide_arv*50

fail = open(failinimi, encoding = "UTF-8")
i=0
print("sobivad lõngad: ")
for rida in fail:
    rida = int(rida)
 #  tokid.append(rida)
    if mitu_grammi(rida) >= vajaminev_longakogus:
        i = i + 1
        print(str(rida) + " tokki = " + str(mitu_grammi(rida)) + " g lõnga")

print("Sobivaid lõngu on " + str(i))
```

0.1 Similarity 100%
Character length: 672

Filename: GG-2161531-assignsubmission-file-/2022-05-27-15-08-59-0.txt;
Filename: BC-2161542-assignsubmission-file-/2022-05-27-14-30-37-0.txt;

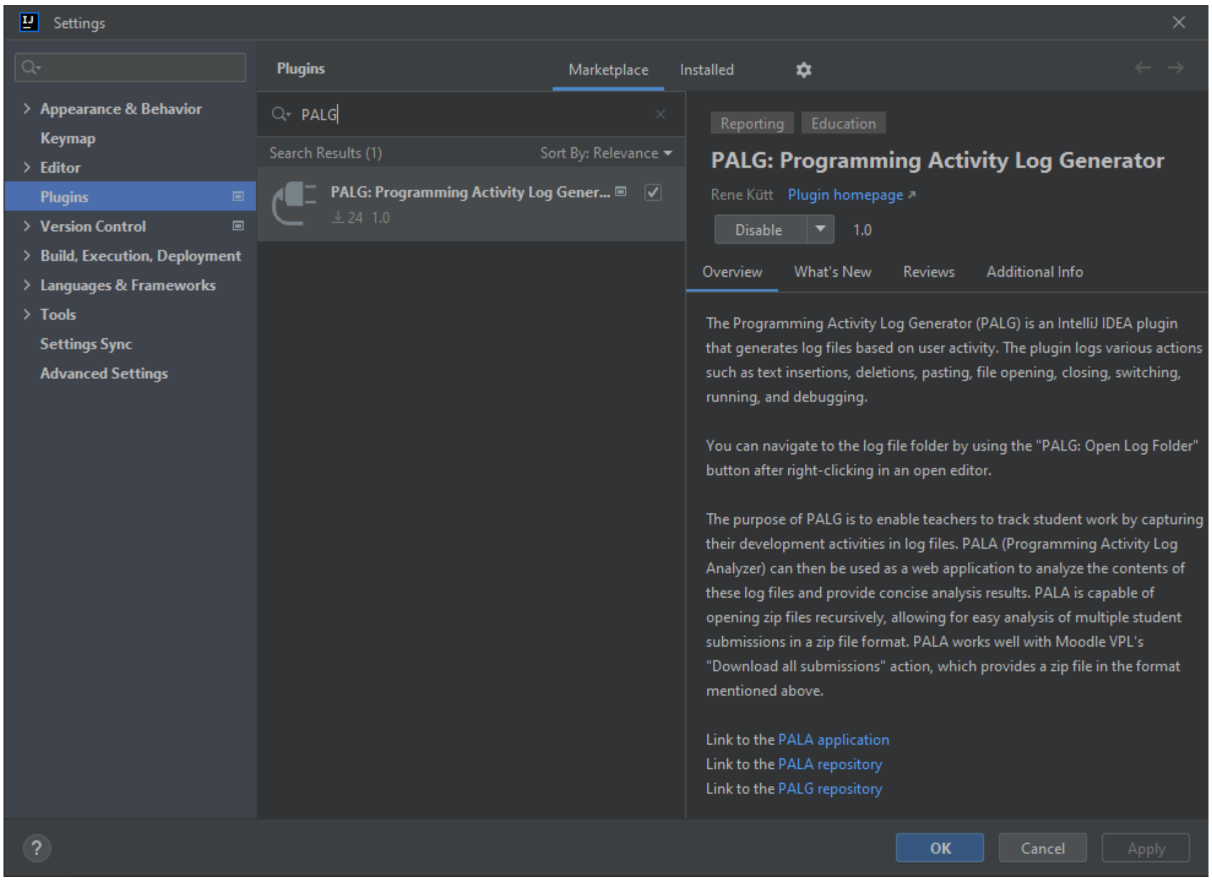
Figure 10. Plagiarism detection source code comparison.

The results of the source code comparison analysis are presented in the sidebar, with list item values showing either log file or folder names depending on the chosen output grouping. In the result window, matched source codes are displayed in separate rows, separated by line separators. Each row displays the similarity percentage and text length at the top, followed by two Bootstrap card elements for the matched source codes. The card header includes the source code file name, log file name, and folder name (if the grouped output was chosen). The left card displays the source code related to the active log file (or folder for grouped output). There is an identification numbering in front of the similarity percentage at the top. The first identification number changes when a different source code finding is displayed. The second number changes if the previous source code is the same and resets when a different source code is compared.

The benefit of the source code comparison analysis is that it helps identify potential instances of plagiarism in student work. This analysis can save teachers time by quickly identifying potential instances of plagiarism, allowing them to focus on reviewing those specific sections of the student work in more detail. Additionally, it can serve as a deterrent to students who might be tempted to plagiarise, as they know their work will be thoroughly checked.

6.2 PALG

The created IntelliJ Platform plugin was published to the JetBrains Marketplace[12]. The version control system used throughout the development process was GitHub[13]. To download the IntelliJ platform plugin PALG, users can follow the installation instructions on the official plugin page[14] or use the JetBrains Plugin Marketplace integrated into the IntelliJ Platform IDEs (Figure 11).



PALG: Programming Activity Log Generator

The Programming Activity Log Generator (PALG) is an IntelliJ IDEA plugin that generates log files based on user activity. The plugin logs various actions such as text insertions, deletions, pasting, file opening, closing, switching, running, and debugging.

You can navigate to the log file folder by using the "PALG: Open Log Folder" button after right-clicking in an open editor.

The purpose of PALG is to enable teachers to track student work by capturing their development activities in log files. PALA (Programming Activity Log Analyzer) can then be used as a web application to analyze the contents of these log files and provide concise analysis results. PALA is capable of opening zip files recursively, allowing for easy analysis of multiple student submissions in a zip file format. PALA works well with Moodle VPL's "Download all submissions" action, which provides a zip file in the format mentioned above.

Link to the PALA application
Link to the PALA repository
Link to the PALG repository

Figure 11. PALG plugin listed in the integrated JetBrains Plugin Marketplace.

The PALG plugin is compatible with the following IDEs:

- Android Studio
- AppCode
- CLion

[12] https://plugins.jetbrains.com/plugin/21567-palg-programming-activity-log-generator

[13] https://github.com/Programming-Activity-Log-Analyser/PALG

[14] https://www.jetbrains.com/help/idea/managing-plugins.html

- DataGrip
- DataSpell
- GoLand
- IntelliJ IDEA
- MPS
- PhpStorm
- PyCharm
- Rider
- RubyMine
- WebStorm

The PALG plugin starts logging immediately after installing the plugin with no restart required. The log files are saved into the java.io.tmpdir subfolder named PALG. The directory where log files are saved can be easily navigated by right-clicking in an open editor and clicking the PALG: Open Log Folder button (Figure 12).

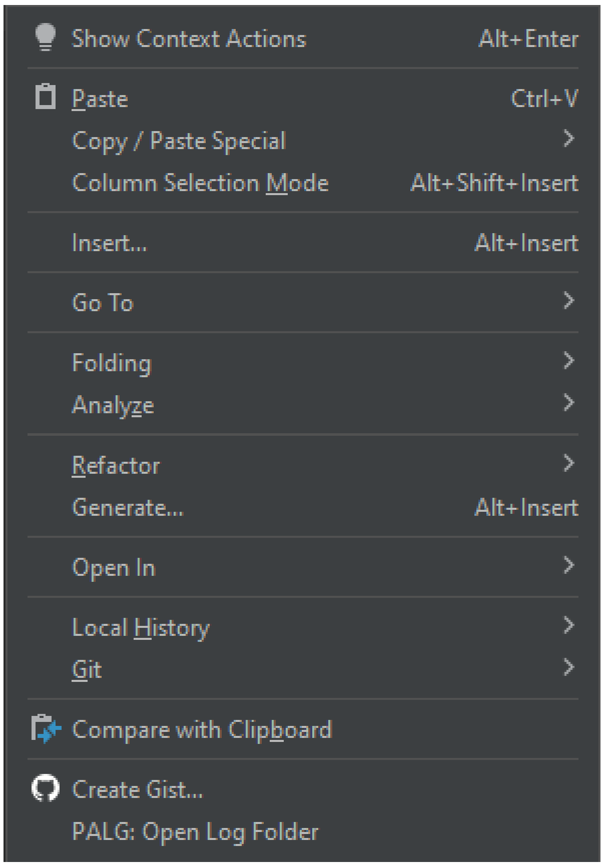
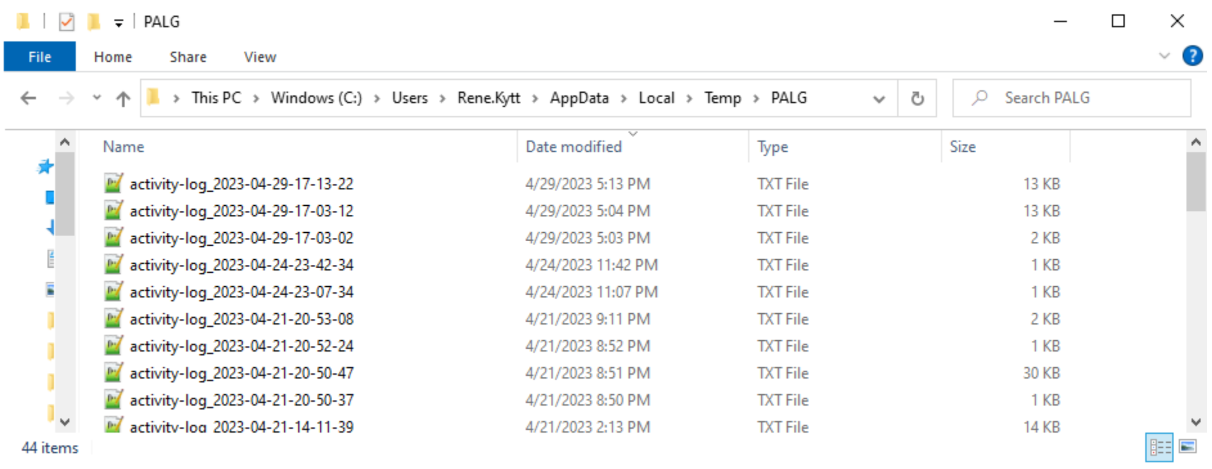


Figure 12. IntelliJ editor popup menu with PALG: Open Log Folder link.

This will open the File Explorer with either the java.io.tmpdir folder with the PALG subfolder active or directly the PALG subfolder (Figure 13). This depends on the IDE and the version of the IDE used.

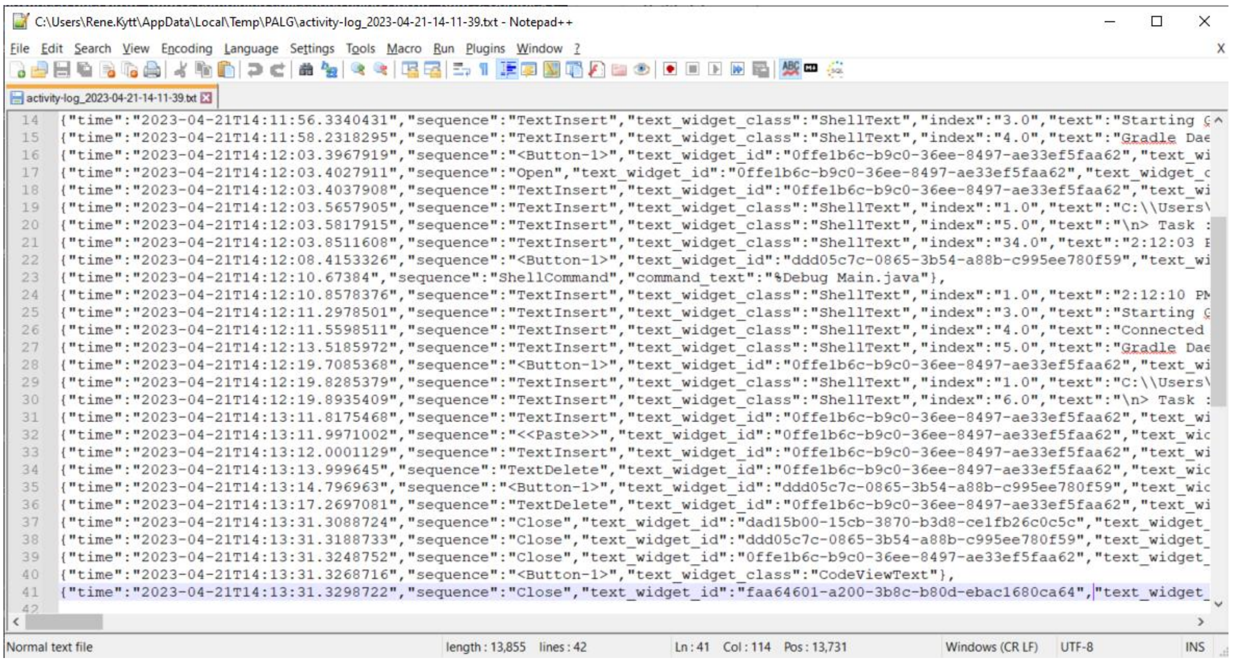

PALG
File
Home
Share
View
This PC › Windows (C:) › Users › Rene.Kytt › AppData › Local › Temp › PALG
Search PALG
44 items

```
Name                              Date modified        Type       Size
activity-log_2023-04-29-17-13-22  4/29/2023 5:13 PM    TXT File   13 KB
activity-log_2023-04-29-17-03-12  4/29/2023 5:04 PM    TXT File   13 KB
activity-log_2023-04-29-17-03-02  4/29/2023 5:03 PM    TXT File    2 KB
activity-log_2023-04-24-23-42-34  4/24/2023 11:42 PM   TXT File    1 KB
activity-log_2023-04-24-23-07-34  4/24/2023 11:07 PM   TXT File    1 KB
activity-log_2023-04-21-20-53-08  4/21/2023 9:11 PM    TXT File    2 KB
activity-log_2023-04-21-20-52-24  4/21/2023 8:52 PM    TXT File    1 KB
activity-log_2023-04-21-20-50-47  4/21/2023 8:51 PM    TXT File   30 KB
activity-log_2023-04-21-20-50-37  4/21/2023 8:50 PM    TXT File    1 KB
activity-log_2023-04-21-14-11-39  4/21/2023 2:13 PM    TXT File   14 KB
```

Figure 13. PALG logging directory open in File Explorer.

The log file names are formatted with the date and time of creation using the following format: "YYYY-MM-DD-HH-MM-SS" preceded by the "activity-log" descriptor. The purpose of this format is to make it easy to sort and identify log files based on their creation time.

The generated logs contain JSON objects of events captured in the IDE (Figure 14).


C:\Users\Rene.Kytt\AppData\Local\Temp\PALG\activity-log_2023-04-21-14-11-39.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
activity-log_2023-04-21-14-11-39.txt

```
{"time":"2023-04-21T14:11:56.3340431","sequence":"TextInsert","text_widget_class":"ShellText","index":"3.0","text":"Starting G
{"time":"2023-04-21T14:11:58.2318295","sequence":"TextInsert","text_widget_class":"ShellText","index":"4.0","text":"Gradle Dae
{"time":"2023-04-21T14:12:03.3967919","sequence":"<Button-1>","text_widget_id":"0ffe1b6c-b9c0-36ee-8497-ae33ef5faa62","text_wi
{"time":"2023-04-21T14:12:03.4027911","sequence":"Open","text_widget_id":"0ffe1b6c-b9c0-36ee-8497-ae33ef5faa62","text_widget_c
{"time":"2023-04-21T14:12:03.4037908","sequence":"TextInsert","text_widget_id":"0ffe1b6c-b9c0-36ee-8497-ae33ef5faa62","text_wi
{"time":"2023-04-21T14:12:03.5657905","sequence":"TextInsert","text_widget_class":"ShellText","index":"1.0","text":"C:\\Users\
{"time":"2023-04-21T14:12:03.5817915","sequence":"TextInsert","text_widget_class":"ShellText","index":"5.0","text":"\n> Task :
{"time":"2023-04-21T14:12:03.8511608","sequence":"TextInsert","text_widget_class":"ShellText","index":"34.0","text":"2:12:03 E
{"time":"2023-04-21T14:12:08.4153326","sequence":"<Button-1>","text_widget_id":"ddd05c7c-0865-3b54-a88b-c995ee780f59","text_wi
{"time":"2023-04-21T14:12:10.67384","sequence":"ShellCommand","command_text":"%Debug Main.java"},
{"time":"2023-04-21T14:12:10.8578376","sequence":"TextInsert","text_widget_class":"ShellText","index":"1.0","text":"2:12:10 PM
{"time":"2023-04-21T14:12:11.2978501","sequence":"TextInsert","text_widget_class":"ShellText","index":"3.0","text":"Starting G
{"time":"2023-04-21T14:12:11.5598511","sequence":"TextInsert","text_widget_class":"ShellText","index":"4.0","text":"Connected
{"time":"2023-04-21T14:12:13.5185972","sequence":"TextInsert","text_widget_class":"ShellText","index":"5.0","text":"Gradle Dae
{"time":"2023-04-21T14:12:19.7085368","sequence":"<Button-1>","text_widget_id":"0ffe1b6c-b9c0-36ee-8497-ae33ef5faa62","text_wi
{"time":"2023-04-21T14:12:19.8285379","sequence":"TextInsert","text_widget_class":"ShellText","index":"1.0","text":"C:\\Users\
{"time":"2023-04-21T14:12:19.8935409","sequence":"TextInsert","text_widget_class":"ShellText","index":"6.0","text":"\n> Task :
{"time":"2023-04-21T14:13:11.8175468","sequence":"TextInsert","text_widget_id":"0ffe1b6c-b9c0-36ee-8497-ae33ef5faa62","text_wi
{"time":"2023-04-21T14:13:11.9971002","sequence":"<<Paste>>","text_widget_id":"0ffe1b6c-b9c0-36ee-8497-ae33ef5faa62","text_wic
{"time":"2023-04-21T14:13:12.0001129","sequence":"TextInsert","text_widget_id":"0ffe1b6c-b9c0-36ee-8497-ae33ef5faa62","text_wi
{"time":"2023-04-21T14:13:13.999645","sequence":"TextDelete","text_widget_id":"0ffe1b6c-b9c0-36ee-8497-ae33ef5faa62","text_wic
{"time":"2023-04-21T14:13:14.796963","sequence":"<Button-1>","text_widget_id":"ddd05c7c-0865-3b54-a88b-c995ee780f59","text_wic
{"time":"2023-04-21T14:13:17.2697081","sequence":"TextDelete","text_widget_id":"0ffe1b6c-b9c0-36ee-8497-ae33ef5faa62","text_wi
{"time":"2023-04-21T14:13:31.3088724","sequence":"Close","text_widget_id":"dad15b00-15cb-3870-b3d8-ce1fb26c0c5c","text_widget_
{"time":"2023-04-21T14:13:31.3188733","sequence":"Close","text_widget_id":"ddd05c7c-0865-3b54-a88b-c995ee780f59","text_widget_
{"time":"2023-04-21T14:13:31.3248752","sequence":"Close","text_widget_id":"0ffe1b6c-b9c0-36ee-8497-ae33ef5faa62","text_widget_
{"time":"2023-04-21T14:13:31.3268716","sequence":"<Button-1>","text_widget_class":"CodeViewText"},
{"time":"2023-04-21T14:13:31.3298722","sequence":"Close","text_widget_id":"faa64601-a200-3b8c-b80d-ebac1680ca64","text_widget_
```

Figure 14. Example contents of log file generated by PALG.

The resulting log file is not valid JSON and needs to be altered by removing the last comma and wrapping the objects in brackets. This was done to keep the program as simple as possible and may be changed in the future if necessary.

Every logged JSON object contains a time attribute and a sequence attribute, which track when an event took place and what event took place, respectively. Additionally, JSON objects have event-specific attributes to describe the specific type of event. For example, the TextInsert event logs where the event took place (program file or shell text), the program file id (if it took place in a program file), the position index it took place in the text, and the inserted text itself. The logged objects contain all the values Thonny Log Analyser uses for analysing. Some property values and object types Thonny IDE logs are omitted as they are unnecessary for the analysis.

6.3 PALA

The created plagiarism detection tool can be accessed through the following link: https://kodu.ut.ee/~kuttrene/PALA/. The version control system used throughout the development process was GitHub[15]. The PALA application is a modified version of the Thonny Log Analyser project that retains all its features and appearance. However, it has been specifically adapted to handle log files generated by the PALG plugin. This required converting the contents of the PALG log files into valid JSON format, as well as updating the event processing logic to be compatible with the structure of the PALG-generated logs. For instance, the PALG logs include a "pasted" event before the "text insertion" event to indicate when the text was pasted, which is the opposite of the order in the Thonny Log Analyser. This change was necessary because of differences in available listeners within the IntelliJ platform.

[15] https://github.com/Programming-Activity-Log-Analyser/PALA

7 Validation

This chapter presents the validation of the plagiarism detection tool, the PALG plugin and the PALA application. Additionally, a comparison with existing solutions and usability testing with teachers is presented.

All of the programs in this thesis have undergone thorough validation to ensure that it meets all the functional and non-functional requirements. The author of this thesis conducted tests to verify that all the requirements were met, and the thesis supervisors further validated the functional requirements.

7.1 Validation of the Plagiarism Detection Tool

The tool was tested for its ability to integrate with Thonny Log Analyser, compare log files, analyse log files for various metrics, and provide quick overviews to teachers. The non-functional requirements, such as user-friendliness, stability, and compatibility with different browsers, were also validated.

To test the non-functional requirements: NFReq.1, NFReq.2, and NFReq.3, usability testing was carried out by the supervisors. The results were that the plagiarism tool has a user-friendly interface that is easy to navigate, provides clear and concise results, and the solution is stable and reliable, thus validating that the requirements have been met.

During the testing phase, the author analysed a ZIP file with a size of 9.24MB to verify whether the plagiarism detection tool met the non-functional requirement of analysing a ZIP file of 10MB or less in less than 30 seconds. The test was conducted three times, and the average time for plagiarism analysis was approximately 11 seconds, which is well within the specified time limit. Therefore, it can be concluded that the plagiarism detection tool meets the non-functional requirement NFReq.4 of analysing a ZIP file with a size of 10MB or less for plagiarism in less than 30 seconds.

The author performed compatibility testing on the plagiarism detection tool using the latest versions of popular web browsers, including Google Chrome version 112.0.5615.183, Microsoft Edge version 113.0.1774.35, and Firefox version 112.0.2. This ensures that it can be used on a wide range of platforms and validates the NFReq.5 requirement.

The validation results indicate that the plagiarism detection tool successfully meets all the requirements and is a reliable and effective solution for detecting plagiarism in student work.

7.2 Validation of PALG

The plugin was tested for compatibility with the latest versions of IntelliJ Platform IDEs, and it was found to function seamlessly with the IDEs. The versions of the IDEs tested were IntelliJ IDEA 2023.1 and PyCharm 2023.1. Additionally, the plugin was tested for logging user actions such as text typed, text pasted, text deleted, file ran, file debugged, file switched, file opened, and file closed. For all tested actions the plugin generated JSON log files in a format that is compatible with Thonny IDE's generated logs.

Moreover, the plugin was evaluated for its non-functional requirements, including ease of installation and configuration, adherence to best practices for IntelliJ IDEA plugin development, and its impact on the performance of IntelliJ Platform IDEs.

The installation and configuration process for the plugin was made effortless with the integration of JetBrains Marketplace into IntelliJ Platform IDEs. As a result, users can quickly and easily install the PALG plugin with just a few clicks, allowing them immediately to start logging user actions and generating log files.

Adherence to best practices for IntelliJ Platform plugin development was thoroughly analysed using SonarLint[16]. The results showed zero warnings, indicating a high level of code quality. Moreover, the plugin was developed in compliance with all the guidelines for plugin development recommended by JetBrains[17], further attesting to its reliability and effectiveness.

The performance of the PALG plugin was evaluated using IntelliJ Analyse Startup Performance and JProfiler[18]. The results of the startup performance analysis using IntelliJ Analyse Startup Performance showed an average startup time of 175ms on three tests. This is considered a good value, indicating that the plugin does not significantly affect the startup time of the IntelliJ Platform IDEs.

JProfiler was used to perform a more in-depth performance analysis of the PALG plugin. The tool was used to measure the CPU processing power and memory usage of the plugin. The results of the CPU profiling showed that the PALG plugin used less than 0.05% of the CPU processing power when changes were made in the IDE, which activated the plugin's logging.

[16] https://www.sonarsource.com/products/sonarlint/
[17] https://plugins.jetbrains.com/docs/intellij/about.html
[18] https://www.ej-technologies.com/products/jprofiler/overview.html

This indicates that the plugin has minimal impact on the overall performance of the IntelliJ Platform IDEs.

Memory profiling was also performed using JProfiler, and the results showed no memory leaks caused by the PALG plugin. This indicates that the plugin is well-written and does not cause any memory-related issues.

In conclusion, the validation results indicate that the PALG plugin is a reliable and effective solution for logging user actions in IntelliJ Platform IDEs and generating JSON log files for plagiarism detection. The plugin successfully meets all functional and non-functional requirements.

7.3 Validation of PALA

The PALA application was tested for its ability to analyse log files created by the PALG IntelliJ plugin and provide the same functionalities as Thonny Log Analyser, including the added plagiarism detection tool. These functional requirements were validated through manual testing.

The validation results indicate that PALA successfully meets all the requirements and is a reliable and effective tool for analysing log files created by the PALG IntelliJ plugin. In addition, the plagiarism detection tool integrated into PALA provides an additional layer of functionality for teachers to detect plagiarism in student work.

7.4 Comparison with Existing Solutions

Among the identified solutions, only MOSS-TAPS, ShowYourWork, and Vipassana use a history-based algorithm, which is also used in the plagiarism detection tool developed in this thesis. MOSS-TAPS and Vipassana have similarities in their approach to detecting plagiarism by utilising repositories to obtain a timeline overview of changes made and using MOSS for analysis. However, MOSS-TAPS and Vipassana are susceptible to obfuscation methods due to their reliance on source-code analysis, which is more advanced than the plain text Dice's coefficient calculation used in this thesis. Despite this, the developed plagiarism detection tool compares all source codes created during the development, not just the ones uploaded to the repository. In addition, it includes a variety of plagiarism analysis types not found in MOSS-TAPS or Vipassana, like duplicate files analysis, student work analysis, pasted to typed texts analysis, identical pasted texts analysis, and source code pasted analysis. ShowYourWork produces log files similar to those generated by Thonny IDE and PALG but has limited functionality in analysing them, being restricted to manually replaying files for finding plagiarism cases. This functionality was already present in Thonny Log Analyser before this thesis and was greatly expanded with the plagiarism detection tool. Overall, the developed plagiarism detection tool provides a new and innovative solution for plagiarism detection.

7.5 Usability Testing with the Supervisors

As part of the development process, the thesis supervisors performed usability tests and provided feedback to improve the user interface design of the plagiarism detection tool. The teachers used the tool in three of their courses where Thonny Log Analyser was already used.

Based on the feedback received from the teachers, several improvements were made to the user interface design of the plagiarism detection tool. The changes included simplifying the layout of the tool, improving the labelling of features, and adding more intuitive visual cues.

In addition to usability testing, the supervisors also provided feedback on the default plagiarism analysis metrics used by the tool. Based on their input, the default metrics were adjusted to better align with the needs of teachers and their expectations for detecting plagiarism. For example, the default source code similarity percentage metric was changed from 99% to 90% so that the system would display more potential plagiarism cases by default.

The supervisors' feedback was crucial in developing the plagiarism detection tool. It ensured that the tool met the requirements of educators and was user-friendly.

8 Possibilities for Future Work

In terms of future work, there are various possibilities for improving the plagiarism detection tool. It could be enhanced with additional analysis types to provide a more detailed comparison of students' development processes and detect plagiarism more effectively, for example, a timeline analysis which would compare students' development timelines. The source code comparison analysis type could be updated to include language-specific similarity algorithms.

Moreover, the PALG and PALA applications can be improved by providing a replayer with a more native look, which displays similar data as that found in the IntelliJ Platform IDEs, such as the entire project structure. Additionally, the applications can be updated to incorporate IntelliJ Platform IDE capabilities, such as auto-complete, to provide a more accurate representation of the code written. Eliminate the chances of false positive plagiarism results due to boilerplate code present in some programming languages or auto-complete being used extensively. Additionally, PALA and Thonny Log Analyser could be linked to make it easier for educators to move between applications.

Conclusions

In conclusion, this thesis has presented a comprehensive solution for detecting plagiarism in programming assignments through the development and implementation of a plagiarism detection tool. The tool integrates with the Thonny Log Analyser web application and uses a history-based approach to monitor students' progress during development and compare it to others. The PALG (Programming Activity Log Generator) plugin and PALA (Programming Activity Log Analyser) application were created to expand the scope of the solution from being able to analyse Thonny IDE logs to be able to generate logs for IntelliJ Platform IDEs and analyse them. The methodology used for software development has been described in detail. The thesis has outlined the functional and non-functional requirements of the plagiarism detection tool, as well as the technical aspects and functionalities of both the plagiarism detection tool, the PALG plugin and the PALA application.

The results obtained from the implementation have been presented, and the solutions have been validated, including a comparison with existing solutions and usability testing with the supervisors. The thesis has also discussed the possibilities for future work, highlighting potential areas for improvement and expansion of the developed solution.

Overall, the developed solution is expected to be a valuable tool for educators in detecting potential plagiarism in programming assignments. The solutions outlined in this thesis provide a seamless and efficient solution for plagiarism detection that addresses the limitations of existing solutions.

References

[1] Kütt, R. (2021). Thonny logide analüüsi veebirakendus (Bachelor's thesis), University of Tartu, Institute of Computer Science. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=71885&year=2021 (09.05.2023)

[2] 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,* 19(3), 1–37. https://doi.org/10.1145/3313290

[3] Sheahen, D., & Joyner, D. (2016). TAPS: A MOSS Extension for Detecting Software Plagiarism at Scale. *Proceedings of the Third (2016) ACM Conference on Learning @ Scale.* Presented at the L@S 2016: Third (2016) ACM Conference on Learning @ Scale, Edinburgh Scotland UK. https://doi.org/10.1145/2876034.2893435

[4] Cheers, H., Lin, Y., & Smith, S. P. (2020, February 3). Detecting pervasive source code plagiarism through dynamic program behaviours. *Proceedings of the Twenty-Second Australasian Computing Education Conference.* https://doi.org/10.1145/3373165.3373168

[5] Blanchard, J., Hott, J. R., Berry, V., Carroll, R., Edmison, B., Glassey, R., … Russell, S. (2022). Stop reinventing the wheel! Promoting community software in computing education. *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education.* https://doi.org/10.1145/3571785.3574129

[6] Rodríguez-del-Pino, J.C., Royo, E.R., & Figueroa, Z.J. (2012). A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. *Proceedings of the 2012 International Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government.* July 2012, Las Vegas, USA.

[7] Schleimer, S., Wilkerson, D. S., & Aiken, A. (2003, June 9). Winnowing. *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data.* https://doi.org/10.1145/872757.872770

[8] Hart, K., Mano, C., & Edwards, J. (2023, March 2). Plagiarism deterrence in CS1 through keystroke data. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education* V. 1. https://doi.org/10.1145/3545945.3569805

[9] Nuzhat, N. (2018). Show Your Work JS. GitHub. https://github.com/nnuzaba/Show-Your-Work-JS/tree/master/Replay-Highlight (09.05.2023)

[10] Koss, I., & Ford, R. (2013). Authorship is continuous: Managing code plagiarism. *IEEE Security & Privacy,* 11(2), 72–74. https://doi.org/10.1109/msp.2013.26

Appendix

I. License