Web-Based Source Code Plagiarism Detection Application Using the Rabin-Karp Algorithm

Ahmad Syarif[1], Samsudin[2], Triase[3]
Information Systems / SAINTEK, Universitas Islam Negeri Sumatera Utara

Source code plagiarism has become a recurring issue in higher education, particularly in programming-related courses where students may copy portions or entire segments of source code from their peers. This situation creates challenges for lecturers in evaluating the originality of student assignments, especially when dealing with a large number of submissions within limited time constraints. Therefore, this study aims to design and develop a web-based source code plagiarism detection application using the Rabin–Karp algorithm to identify and measure the similarity level between programming code documents. The research employed a qualitative approach through observation, interviews, and literature review, while system development followed the Agile methodology. The developed application was tested using source code files written in C++, Java, and Python programming languages. Black-box testing demonstrated that all system functions operated successfully, including file uploading, preprocessing, tokenization, rolling hash generation, fingerprint matching, and similarity calculation. The validity testing results showed similarity percentages ranging from 2.81%–62.70% for C++ files, 17.26%–54.49% for Java files, and 6.35%–34.89% for Python files. These findings indicate that the application can effectively detect similarities between source code documents and support lecturers in identifying potential plagiarism cases. Furthermore, the Rabin–Karp algorithm proved capable of performing similarity analysis efficiently across multiple programming languages with relatively fast processing time.

Keywords: Plagiarism, Rabin-Karp, Source Code.

Corresponding Author:
Ahmad Syarif
Information Systems / SAINTEK, Universitas Islam Negeri Sumatera Utara
ahmad.syarif@uinsu.ac.id

1. Introduction

Information and Communication Technology (ICT) has become an inseparable part of the modern world. Cultural and societal aspects must adapt to meet the challenges of the knowledge era. The rapid diffusion of ICT has brought significant changes in social, political, economic, and educational domains [1]). For educators, such as lecturers, plagiarism detection tools can be utilized to identify potential similarities between two or more documents, thereby minimizing the time and effort required. Plagiarism is commonly defined as copying or borrowing someone else's ideas or words and claiming them as one's own. This practice is often found in novels, scientific works, artistic designs, and other forms of intellectual output. However, plagiarism has now extended to program code as well [2].

Code plagiarism represents a significant issue in academic environments, as students often attempt to copy parts or even entire sections of program code from their peers. Currently, several applications are available to automatically detect code plagiarism, one of which is JPlag. JPlag is a web-based application used to detect plagiarism in programming languages such as Java, C, and C++. Each submitted program code is converted into a string of tokens, which are then compared using algorithms such as Greedy String Tiling and Rabin-Karp ([3].

Previous research conducted by Iwan Firmawan, [4], entitled *"Design and Development of a Source Code Plagiarism Detection Application Using JPlag Tools"*, discussed the utilization of JPlag to develop a

plagiarism detection application called Scaniplag. The Faculty of Science and Technology at the State Islamic University (UIN) of North Sumatra was established on Tuesday, December 29, 2015. To date, the faculty has developed five study programs, including the undergraduate Information Systems program. Based on interviews conducted with several lecturers and students of the Information Systems program at UIN North Sumatra, instances of plagiarism particularly code plagiarism are still frequently encountered. One of the main challenges faced by lecturers in detecting code plagiarism is the large number of program documents that must be reviewed within a limited time. Additionally, maintaining accuracy when comparing multiple program codes is also a significant difficulty.

2. Literature Review

Plagiarism detection in source code has attracted significant attention in computer science education due to the increasing number of programming assignments submitted through digital platforms. Source code plagiarism differs from textual plagiarism because students can modify variable names, formatting, comments, or program structures while preserving the same underlying logic. Consequently, conventional text-matching techniques are often insufficient to identify similarities accurately. Previous studies have demonstrated that string-matching algorithms can effectively detect code similarity by analyzing patterns and token sequences within source code. The Rabin–Karp algorithm is widely recognized for its efficiency in multi-pattern matching through hash-based comparisons, making it suitable for plagiarism detection systems. Research by [5] showed that source code similarity detection can be improved through tokenization and string comparison approaches, while [6] highlighted the effectiveness of Rabin–Karp in similarity analysis tasks. Furthermore, [7] reported that Rabin–Karp provides reliable performance in detecting plagiarism in student assignment documents.

Several studies have developed plagiarism detection systems using different approaches and tools. [8] implemented the JPlag tool for detecting plagiarism in programming assignments and demonstrated its usefulness in academic environments. Meanwhile, [9] applied the Rabin–Karp algorithm to compare source code similarity and found that the similarity percentage is influenced by the selected k-gram value. Although previous studies have confirmed the effectiveness of plagiarism detection methods, most existing systems are either dependent on external tools or focus on specific programming languages. In addition, limited research has examined the implementation of a web-based plagiarism detection system that can compare multiple programming languages while providing similarity measurements through an integrated platform. This gap indicates the need for a practical and efficient system capable of assisting lecturers in evaluating programming assignments across different programming environments.Existing approaches often rely on external tools or have limited flexibility in handling multiple programming languages. Therefore, this study seeks to answer the following research questions: How can a web-based application be developed to detect source code plagiarism using the Rabin–Karp algorithm?, How effectively can the Rabin–Karp algorithm measure similarity levels between source code documents written in different programming languages?, Can the developed system assist lecturers in identifying potential plagiarism cases more efficiently than manual inspection methods?

3. Methods

The study employs a qualitative approach through observation, interviews, and literature review, while applying the Agile system development model. The system requirements are supported by hardware such as an Intel Core i3-based laptop and software including Windows 10, Laragon, Sublime Text, and Microsoft Visio. The data utilized consist of primary data (obtained through observation and interviews) and secondary data (derived from literature and previous studies). The system development process follows

Agile stages, including planning, requirements analysis, design (UML, database, and interface), coding, testing (black-box), and implementation and maintenance, with the aim of producing an effective and efficient program code plagiarism detection system

4. Result And Discussion

Design Plan

The program code plagiarism detection application begins with the process of inputting at least two program code document files to analyze their similarity values. The files can only be uploaded in .zip format, which contains the program code files that will be extracted during the upload process. The process primarily emphasizes the use of the Rabin-Karp algorithm to process string similarity, enabling the analysis of plagiarism in program code. The overview of the system process is as follows:

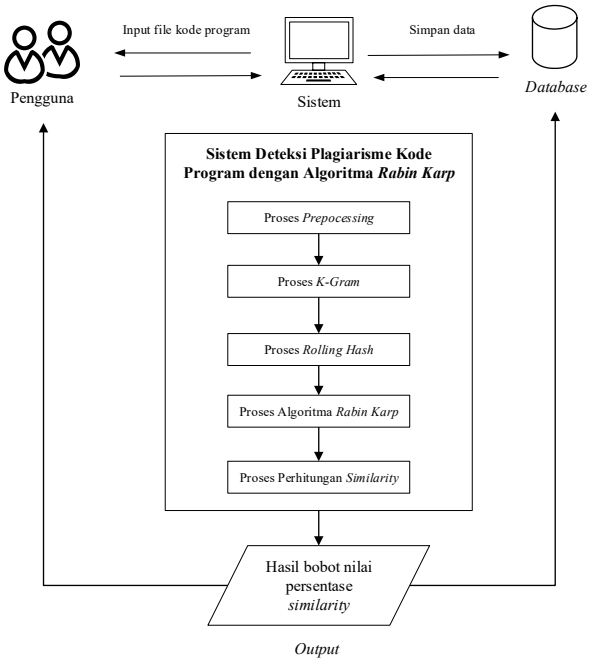


Figure 1. System Process Overview of Program Code Plagiarism Detection Using the Rabin-Karp Algorithm

Process Model Design

Use Case Diagram

The use case diagram is utilized to identify the functions available within the information system and to determine who is authorized to use these functions. It describes the relationship between actors and the system being developed. In this diagram, there are two actors: Admin and User (lecturer). The use case diagram for the program code plagiarism detection application is presented below:

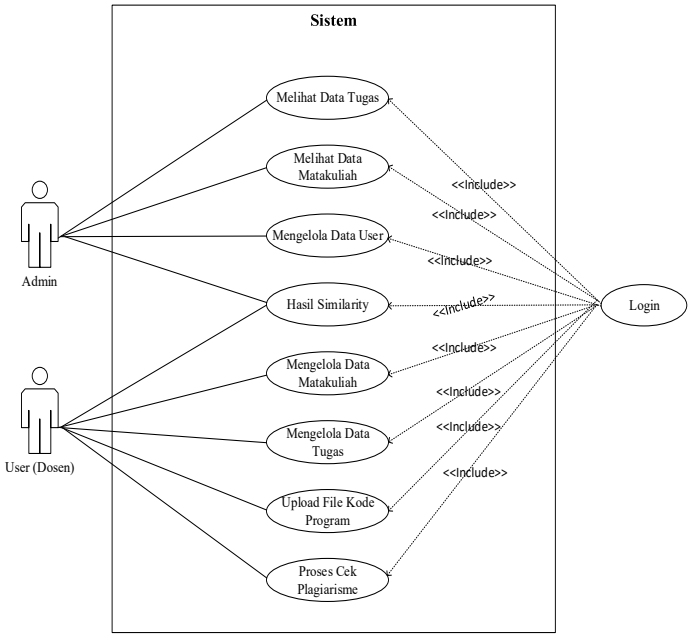


Figure 2. Use Case Diagram of the Program Code Plagiarism Detection Application

Activity Diagram

The activity diagram illustrates the workflow or sequence of activities within a system or business process. The following are the activity diagrams for the program code plagiarism detection application:

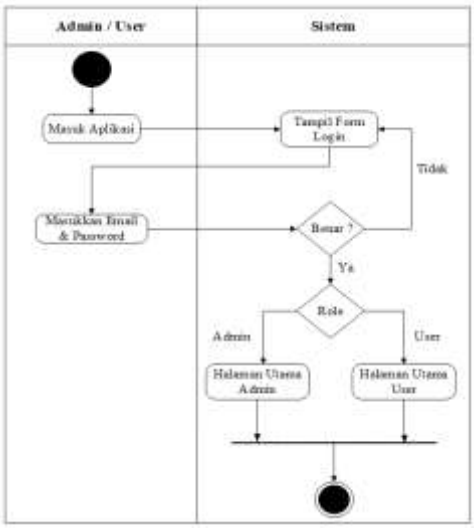
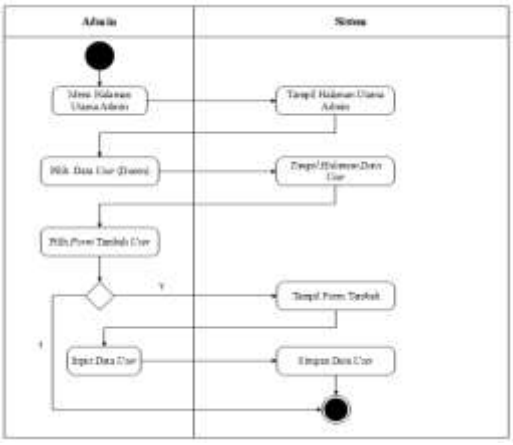


Figure 3. Login Activity Diagram



ISSN 2828-9463

Figure 4. Admin Activity Diagram

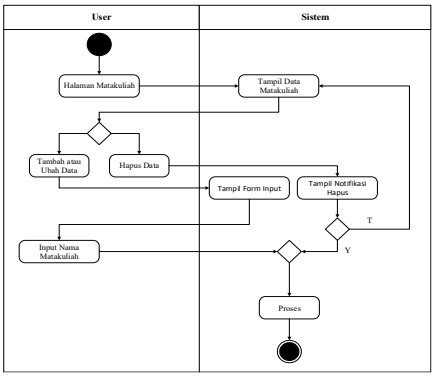


Figure 5. Course Activity Diagram for User Actor

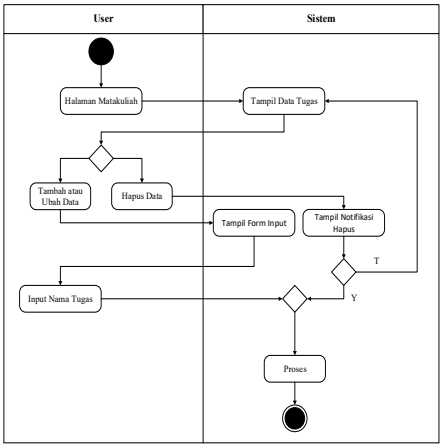


Figure 6. Assignment Activity Diagram for User Actor

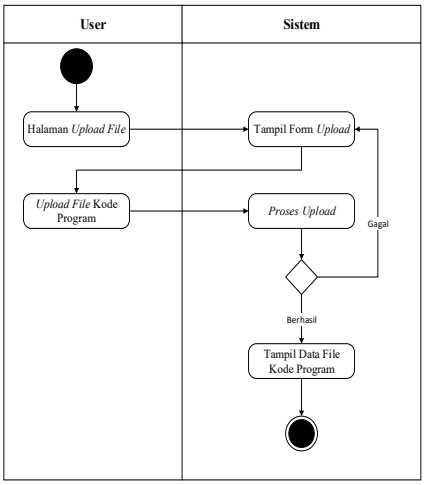


Figure 7. Program Code File Upload Activity Diagram for User Actor

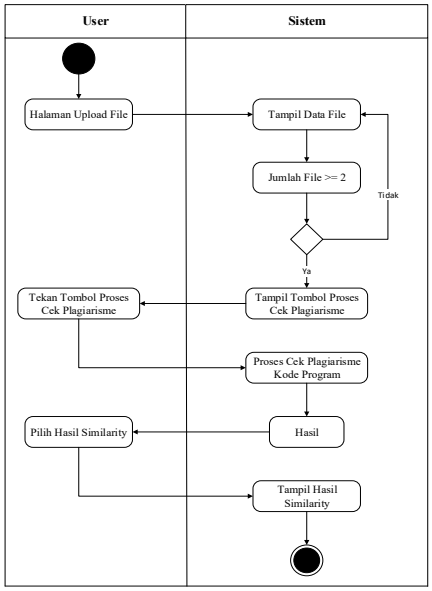


Figure 8. Plagiarism Checking Process Activity Diagram for User Actor

Class Diagram

The class diagram describes the attributes and operations of a class that interacts with other objects having relationships. This diagram supports database development by defining the system structure.

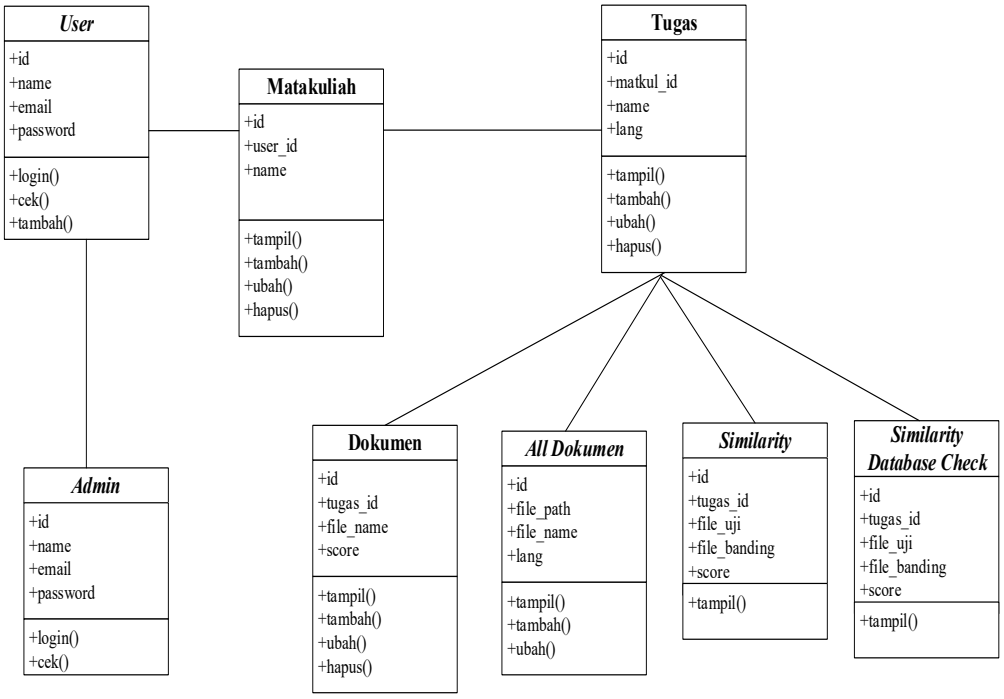


Figure 9. Class Diagram of the Program Code Plagiarism Detection Application

Database Design

Database design is the process of creating a structure that supports the operational needs and development objectives of the application. By utilizing a database, data can be stored, modified, and retrieved efficiently and effectively.

Table Structure Design

Admin Table

This table is used to store admin data, which functions for login/logout activities before accessing the system.

Table 1. Admin Table Structure

No	Field Name	Data Type	Size
1	id	bigint (primary key)	20
2	name	varchar	255
3	email	varchar	255
4	password	varchar	255
5	created_at	timestamp	
6	updated_at	timestamp	

User Table

This table is used to store user data for login/logout activities before accessing the program code plagiarism detection system using the Rabin-Karp algorithm.

Table 2. User Table Structure

No	Field Name	Data Type	Size
1	id	bigint (primary key)	20
2	name	varchar	255
3	email	varchar	255
4	password	varchar	255
5	created_at	timestamp	
6	updated_at	timestamp	

Course Table (Matakuliah)

The course data entered will be stored in this table and can be used to establish relationships with the assignment table.

Table 3. Course Table Structure

No	Field Name	Data Type	Size
1	id	bigint (primary key)	20
2	user_id	int (foreign key)	11
3	name	varchar	255
4	created_at	timestamp	
5	updated_at	timestamp	

Assignment Table (Tugas)

This table contains a field named *lang* to store the programming language data that will be analyzed for plagiarism. The assignment table is related to several other tables.

Table 4. Assignment Table Structure

No	Field Name	Data Type	Size
1	id	bigint (primary key)	20
2	matkul_id	int (foreign key)	11
3	name	varchar	255
4	lang	varchar	255
5	created_at	timestamp	
6	updated_at	timestamp	

Document Table (Dokumen)

This table is used to store document data or program code files that have been uploaded based on the selected assignment and are related to the assignment table.

Table 5. Document Table Structure

No	Field Name	Data Type	Size
1	id	bigint (primary key)	20
2	tugas_id	int (foreign key)	11
3	file_name	varchar	255
4	score	varchar	255
5	created_at	timestamp	
6	updated_at	timestamp	

All Documents Table

Program code files are also stored in this table. While the document table stores files based on assignments, this table stores all program code files to serve as comparison data when users perform global detection.

Table 6. All Documents Table Structure

No	Field Name	Data Type	Size
1	id	bigint (primary key)	20
2	file_path	varchar	255
3	file_name	varchar	255
4	lang	varchar	255
5	created_at	timestamp	
6	updated_at	timestamp	

Similarity Table

This table is used to store the results of program code plagiarism detection using the Rabin-Karp algorithm through iterative comparisons between files.

Table 7. Similarity Table Structure

No	Field Name	Data Type	Size
1	id	bigint (primary key)	20
2	tugas_id	int (foreign key)	11
3	file_uji	varchar	255
4	file_banding	varchar	255
5	score	varchar	255
6	created_at	timestamp	
7	updated_at	timestamp	

Similarity Database Check Table

This table stores the results of plagiarism detection using the Rabin-Karp algorithm through iterative comparisons with previously stored database files.

Table 8. Similarity Database Check Table Structure

No	Field Name	Data Type	Size
1	id	bigint (primary key)	20
2	tugas_id	int (foreign key)	11
3	file_uji	varchar	255

No	Field Name	Data Type	Size
4	file_banding	varchar	255
5	score	varchar	255
6	created_at	timestamp	
7	updated_at	timestamp	

System Interface Design

The interface design is created in a simple manner to facilitate user interaction with the program code plagiarism detection application. The interface consists of both input and output design components.

Login Page Interface Design

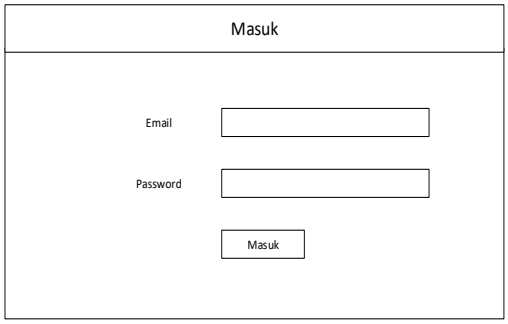


Figure 10. Login Page Interface Design

Home Page Interface Design

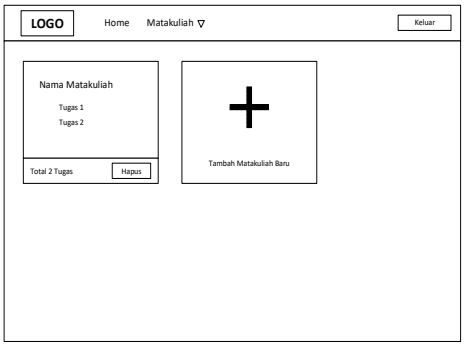


Figure 11. Home Page Interface Design

Course Data Input Interface Design

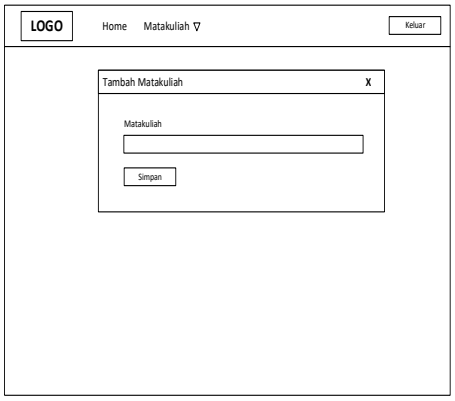


Figure 12. Course Data Input Interface Design

Assignment Data Interface Design

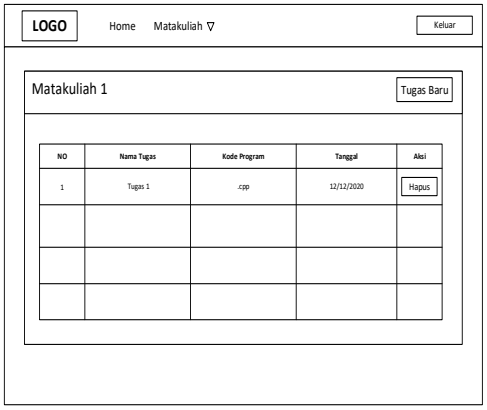


Figure 13. Assignment Data Interface Design

Add Assignment Data Interface Design

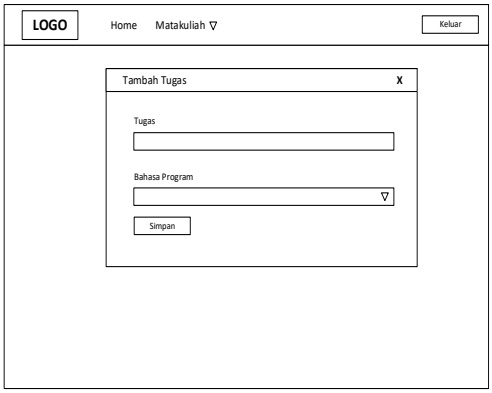


Figure 14. Add Assignment Data Interface Design

Program Code File Upload Interface Design

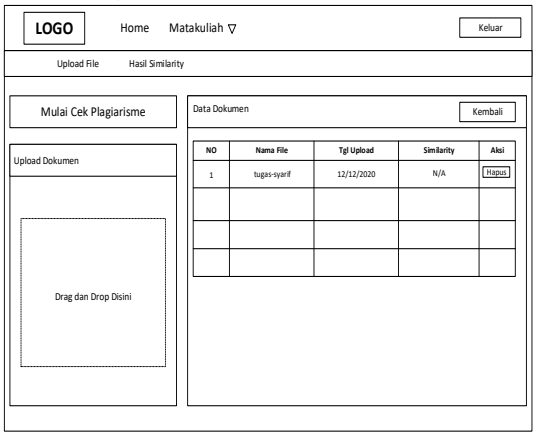


Figure 15. Program Code File Upload Interface Design

Similarity Results Interface Design

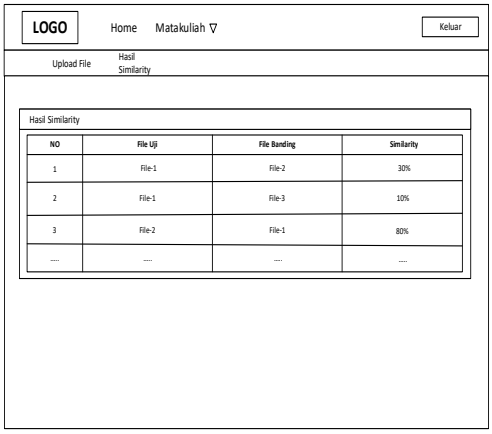


Figure 16. Similarity Results Interface Design

Implementation and Testing

Interface Implementation

The interface implementation of the code plagiarism detection application using the Rabin-Karp algorithm consists of several user interfaces, including a login interface, a main dashboard displaying course data, an interface for adding course data, task data and task creation interfaces, a code file upload interface, and a similarity results interface.

Login Interface

Figure 17. Login Interface

Main Page Interface

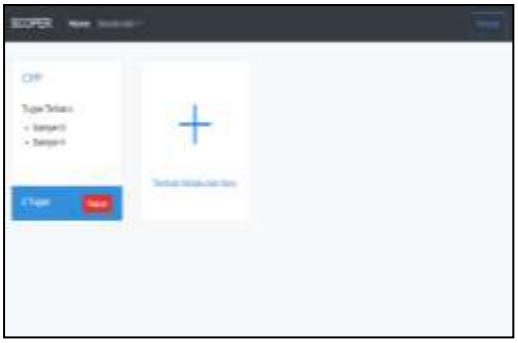


Figure 18. Main Page Interface

ISSN 2828-9463

Figure 19. Add Course Data Interface

Task Data Interface

Figure 20. Task Data Interface

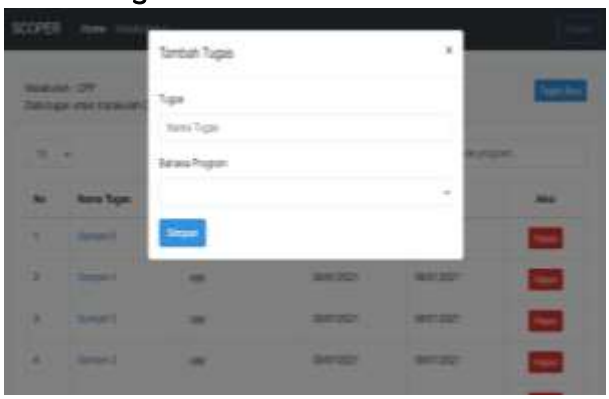


Figure 21. Add Task Interface

Code File Upload Interface

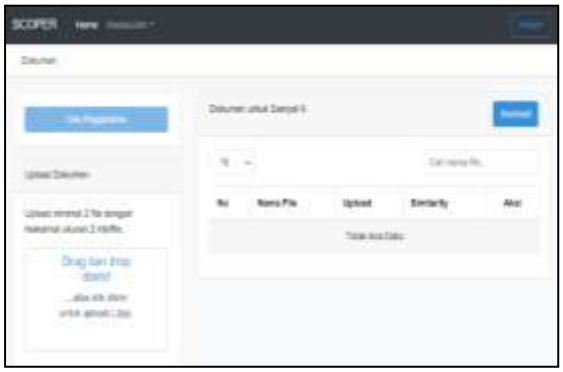


Figure 22. Code File Upload Interface

ISSN 2828-9463

Figure 23. Code File Data Interface

Figure 24. Pop-up Interface for Selecting Plagiarism Checking Method

Similarity Results Interface

Figure 25. Similarity Results Interface

Figure 26. Detailed Process Interface of the Rabin-Karp Algorithm

Testing

System testing aims to ensure the suitability and success of the system in accordance with the objectives established in the study. Black-box testing was conducted by focusing on aspects such as system interface, menu functionalities, and the conformity of functional workflows with the designed model.

Testing Environment

The testing environment includes both hardware and software used during the evaluation of the Word Similarity Detection System, as follows:

1. CPU: Intel(R) Core(TM) i3-5005U @ 2.00 GHz, RAM 2.00 GB
2. Operating System: Windows 10 Home
3. Software: Visual Studio Code 1.52.1, Google Chrome 87.0, XAMPP

Testing Plan

The testing plan for the Code Plagiarism Detection Application is presented in Table 9. All tests were conducted using the black-box method. System validity testing was performed on 5 code files in C++, 5 in Java, and 5 in Python. Each file with similar topics was compared. The data were obtained from programming assignments given to students in semesters 3 and 5.

Table 9. Testing Plan for the Code Plagiarism Detection Application

No	Test Class	Test Item	Testing Type
1	File Upload	Input less than two files; Input two or more files	Black Box
2	Preprocessing Stage	Ensuring removal of comments, headers, and keywords	Black Box
3	K-Gram Stage	Ensuring document parsing into tokens	Black Box
4	Rolling Hash Stage	Ensuring each token produces a hash value	Black Box
5	Fingerprint Stage	Ensuring matched hashes are identical	Black Box
6	Similarity Calculation Stage	Calculating similarity between code files	Black Box

Table 10. Testing Data for C++ Programming Language

No	File Name	∑ Lines	Size
1	file-1.cpp	19 Lines	1 KB

No	File Name	∑ Lines	Size
2	file-2.cpp	57 Lines	1 KB
3	file-3.cpp	28 Lines	1 KB
4	file-4.cpp	18 Lines	1 KB
5	file-5.cpp	80 Lines	2 KB

The C++ files range from 18 to 80 lines, with an average size of 1 KB and a .cpp format. Each file was tested to determine the percentage similarity of the code.

Table 11. Testing Data for Java Programming Language

No	File Name	∑ Lines	Size
1	file-1.java	23 Lines	1 KB
2	file-2.java	15 Lines	1 KB
3	file-3.java	21 Lines	1 KB
4	file-4.java	12 Lines	1 KB
5	file-5.java	13 Lines	1 KB

The Java files range from 12 to 23 lines, with an average size of 1 KB and a .java format. Each file was tested to measure similarity percentages.

Table 12. Testing Data for Python Programming Language

No	File Name	∑ Lines	Size
1	file-1.py	8 Lines	1 KB
2	file-2. py	6 Lines	1 KB
3	file-3. py	19 Lines	1 KB
4	file-4. py	14 Lines	1 KB
5	file-5. py	10 Lines	1 KB

The Python files range from 6 to 19 lines, with an average size of 1 KB and a .py format. Each file was tested to determine similarity percentages.

Testing Results

Table 13. Black Box Testing Results

No	Description	Result	Conclusion
1	Input less than two files	Plagiarism check button inactive	Accepted
2	Input two or more files	1. Button active 2. System reads document content	Accepted
3	Preprocessing removes comments, headers, keywords	Successfully removed	Accepted
4	Document parsing into tokens	System splits into 7 tokens	Accepted
5	Each token produces hash value	Hashing performed using rolling hash	Accepted
6	Matching hashes are identical	Matching performed using Rabin-Karp	Accepted
7	Similarity calculation process	Similarity computed based on document comparison	Accepted

Table 14. Validity Testing Results for C++ Programming Language

No	Test Document		Comparison Document		Similarity
	File Name	∑Lines	File Name	∑Lines	Percentage
1	file-1.cpp	19 Lines	file-2.cpp	57 Lines	2.81%
2	file-1.cpp	19 Lines	file-3.cpp	28 Lines	7.50%

No	Test Document		Comparison Document		Similarity
	File Name	∑Lines	File Name	∑Lines	Percentage
3	file-1.cpp	19 Lines	file-4.cpp	18 Lines	6.31%
4	file-1.cpp	19 Lines	file-5.cpp	80 Lines	3.19%
5	file-2.cpp	57 Lines	file-3.cpp	28 Lines	52.50%
6	file-2.cpp	57 Lines	file-4.cpp	18 Lines	62.70%
7	file-2.cpp	57 Lines	file-5.cpp	80 Lines	49.80%
8	file-3.cpp	28 Lines	file-4.cpp	18 Lines	28.77%
9	file-3.cpp	28 Lines	file-5.cpp	80 Lines	16.18%
10	file-4.cpp	18 Lines	file-5.cpp	80 Lines	26.45%

Plagiarism testing on five C++ source code files was conducted by performing pairwise comparisons, resulting in 10 iterations. The similarity percentage ranged from a minimum of 2.81% to a maximum of 62.70%.

Table 15. Validity Testing Results for Java Programming Language

No	Test Document		Comparison Document		Similarity
	File Name	∑Lines	File Name	∑Lines	Percentage
1	file-1.java	23 Lines	file-2. java	15 Lines	23.13%
2	file-1. java	23 Lines	file-3. java	21 Lines	17.26%
3	file-1. java	23 Lines	file-4. java	12 Lines	27.00%
4	file-1. java	23 Lines	file-5. java	13 Lines	30.67%
5	file-2. java	15 Lines	file-3. java	21 Lines	37.94%
6	file-2. java	15 Lines	file-4. java	12 Lines	46.35%
7	file-2. java	15 Lines	file-5. java	13 Lines	39.35%
8	file-3. java	21 Lines	file-4. java	12 Lines	54.49%
9	file-3. java	21 Lines	file-5. java	13 Lines	43.88%
10	file-4. java	12 Lines	file-5. java	13 Lines	39.17%

Plagiarism testing on five Java source code files was performed through pairwise comparisons, resulting in 10 iterations. The similarity percentage ranged from 17.26% to 54.49%.

Table 16. Validity Testing Results for Python Programming Language

No	Dokumen Uji		Dokumen Banding		Persentase
	Nama File	∑Lines	Nama File	∑Lines	Similarity
1	file-1.py	8 Lines	file-2. py	6 Lines	34.89%
2	file-1.py	8 Lines	file-3. py	19 Lines	30.25%
3	file-1. py	8 Lines	file-4. py	14 Lines	28.84%
4	file-1. py	8 Lines	file-5. py	10 Lines	10.46%
5	file-2. py	6 Lines	file-3. py	19 Lines	28.28%
6	file-2. py	6 Lines	file-4. py	14 Lines	26.11%
7	file-2. py	6 Lines	file-5. py	10 Lines	16.18%
8	file-3. py	19 Lines	file-4. py	14 Lines	9.61%
9	file-3. py	19 Lines	file-5. py	10 Lines	6.35%
10	file-4. py	14 Lines	file-5. py	10 Lines	17.28%

Plagiarism testing on five Python source code files was conducted through pairwise comparisons, resulting in 10 iterations. The similarity percentage ranged from 6.35% to 34.89%.

Discussion

The results indicate that the developed web-based plagiarism detection application successfully implemented the Rabin–Karp algorithm to identify similarities among source code files written in C++, Java, and Python. The black-box testing results confirmed that all system functionalities, including preprocessing, tokenization, rolling hash generation, fingerprint matching, and similarity calculation, operated as expected. These findings support the theoretical principle of the Rabin–Karp algorithm, which utilizes hash values to efficiently identify matching string patterns within large datasets. By converting source code segments into hash representations, the system can reduce computational complexity while maintaining accurate similarity detection. This characteristic makes Rabin–Karp particularly suitable for plagiarism detection applications involving multiple source code files [10].

The validity testing results demonstrated varying similarity percentages across different programming languages. In the C++ dataset, similarity values ranged from 2.81% to 62.70%, while Java and Python datasets produced ranges of 17.26%–54.49% and 6.35%–34.89%, respectively. These variations indicate that similarity scores are influenced by coding structure, programming syntax, and the extent of code reuse among the tested files. The findings are consistent with the study conducted by [2], which reported that similarity measurements generated by the Rabin–Karp algorithm are affected by parameter settings such as k-gram values. Likewise, the results support the work of Leonardo and [11], who concluded that Rabin–Karp is effective for similarity detection because it can identify matching patterns even when documents contain structural differences.

Furthermore, the successful implementation of the system aligns with the findings of [12], who demonstrated that the Rabin–Karp algorithm can provide reliable plagiarism detection performance in academic environments. The developed application extends previous research by integrating plagiarism detection into a web-based platform capable of handling multiple programming languages within a single system. Compared with manual inspection methods, the proposed system offers greater efficiency by automatically processing source code files and generating similarity percentages in a relatively short time [13]. This capability is particularly beneficial for lecturers who must evaluate large volumes of programming assignments while maintaining consistency and objectivity in plagiarism assessment [14].

Overall, the results confirm that the Rabin–Karp algorithm provides an effective approach for source code plagiarism detection. The combination of preprocessing, tokenization, rolling hash, and similarity analysis enables the system to identify potential [15] plagiarism cases accurately while reducing the time required for manual comparison. Therefore, the developed application can serve as a practical tool to support academic integrity and improve the evaluation process of programming assignments in higher education institutions [16].

5. Conclusion

Based on the research that has been conducted, several conclusions can be drawn:

1. The development of a web-based program code plagiarism detection application using the Rabin-Karp algorithm has been successfully implemented. This system is capable of generating a percentage weight of similarity between program codes.
2. The smaller the k-gram value used, the higher the resulting similarity percentage weight; conversely, the larger the k-gram value, the lower the similarity percentage weight obtained.
3. This program code plagiarism detection application is capable of comparing program files written in C++, Java, and Python programming languages.

4. The application is able to perform comparisons between uploaded program files as well as with previously uploaded files, which serve as reference data for similarity detection.
5. The web-based program code plagiarism detection application using the Rabin-Karp algorithm is able to calculate the similarity percentage weight efficiently within a relatively short processing time.

Recommendations

This study has several limitations, including the use of a limited number of source code files and testing that only involved C++, Java, and Python programming languages. In addition, the Rabin–Karp algorithm focuses on string pattern matching, which may be less effective in detecting plagiarism involving significant code restructuring or logic modifications.

Therefore, future research is recommended to utilize larger and more diverse datasets, evaluate additional programming languages, and compare the Rabin–Karp algorithm with other plagiarism detection methods. Further development may also incorporate semantic and structural code analysis to improve detection accuracy and enhance the system's applicability in academic environments.

6. References

[1] B. Simões, M. Del P. Carretero, J. Martínez, S. Muñoz, And N. Alcain, "Implementing Digital Twins Via Micro-Frontends, Micro-Services, And Web 3d," *Comput. Graph.*, Vol. 121, P. 103946, Jun. 2024, Doi: 10.1016/J.Cag.2024.103946.

[2] R. Perlin, D. Ebling, V. Maran, G. Descovi, And A. Machado, "An Approach To Follow Microservices Principles In Frontend," In *2023 Ieee 17th International Conference On Application Of Information And Communication Technologies (Aict)*, Ieee, Oct. 2023, Pp. 1–6. Doi: 10.1109/Aict59525.2023.10313208.

[3] M. Mammetmyradov, N. Faizah, And L. Koryanto, "Aplikasi Pencarian Showroom Yamaha Di Kota Tasikmalaya Berbasis Android Menggunakan Metode Location-Based Service (Lbs) Dan Framework React Native," *Journal Digital Technology Trend*, Vol. 1, No. 2, Pp. 92–98, Dec. 2022, Doi: 10.56347/Jdtt.V1i2.69.

[4] W. Gan, Z. Ye, S. Wan, And P. S. Yu, "Web 3.0: The Future Of Internet," In *Companion Proceedings Of The Acm Web Conference 2023*, New York, Ny, Usa: Acm, Apr. 2023, Pp. 1266–1275. Doi: 10.1145/3543873.3587583.

[5] R. Hadi, S. Melumad, And E. S. Park, "The Metaverse: A New Digital Frontier For Consumer Behavior," *Journal Of Consumer Psychology*, Vol. 34, No. 1, Pp. 142–166, Jan. 2024, Doi: 10.1002/Jcpy.1356.

[6] M. Ali *Et Al.*, "A Simple And Secure Reformation-Based Password Scheme," *Ieee Access*, Vol. 9, Pp. 11655–11674, 2021, Doi: 10.1109/Access.2020.3049052.

[7] B. Leonardo And S. Hansun, "Text Documents Plagiarism Detection Using Rabin-Karp And Jaro-Winkler Distance Algorithms," *Indonesian Journal Of Electrical Engineering And Computer Science*, Vol. 5, No. 2, P. 462, Feb. 2017, Doi: 10.11591/Ijeecs.V5.I2.Pp462-471.

[8] C. Wang, Z. Pei, S. Qiu, And Z. Tang, "Rgb-D-Based Stair Detection And Estimation Using Deep Learning," *Sensors*, Vol. 23, No. 4, P. 2175, Feb. 2023, Doi: 10.3390/S23042175.

[9] C. Kustanto And I. Liem, "Automatic Source Code Plagiarism Detection," In *2009 10th Acis International Conference On Software Engineering, Artificial Intelligences, Networking And Parallel/Distributed Computing*, Ieee, May 2009, Pp. 481–486. Doi: 10.1109/Snpd.2009.62.

[10] Z. Duric And D. Gasevic, "A Source Code Similarity System For Plagiarism Detection," *Comput. J.*, Vol. 56, No. 1, Pp. 70–86, Jan. 2013, Doi: 10.1093/Comjnl/Bxs018.

[11] S. D. Purnamasari And F. Panjaitan, "Pemodelan Sistem Informasi Sebaran Pasar Menggunakan Unified Modeling Language," *Jipi (Jurnal Ilmiah Penelitian Dan Pembelajaran Informatika)*, Vol. 4, No. 2, P. 103, Dec. 2019, Doi: 10.29100/Jipi.V4i2.1402.

[12] M. De Ryck, M. Versteyhe, And F. Debrouwere, "Automated Guided Vehicle Systems, State-Of-The-Art Control Algorithms And Techniques," *J. Manuf. Syst.*, Vol. 54, Pp. 152–173, Jan. 2020, Doi: 10.1016/J.Jmsy.2019.12.002.

[13] A. Filcha And M. Hayaty, "Implementasi Algoritma Rabin-Karp Untuk Pendeteksi Plagiarisme Pada Dokumen Tugas Mahasiswa," *Juita : Jurnal Informatika*, Vol. 7, No. 1, P. 25, May 2019, Doi: 10.30595/Juita.V7i1.4063.

[14] N. Gupta, V. Gandhi, C. Hariya, And V. Shelke, "Detection Of Code Clones," In *2018 International Conference On Smart City And Emerging Technology (Icscet)*, Ieee, Jan. 2018, Pp. 1–4. Doi: 10.1109/Icscet.2018.8537249.

[15] X. Fan, *Real-Time Embedded Systems*. Elsevier, 2015. Doi: 10.1016/C2014-0-00053-6.

[16] S. Guarnieri, M. Pistoia, O. Tripp, J. Dolby, S. Teilhet, And R. Berg, "Saving The World Wide Web From Vulnerable Javascript," In *Proceedings Of The 2011 International Symposium On Software Testing And Analysis*, New York, Ny, Usa: Acm, Jul. 2011, Pp. 177–187. Doi: 10.1145/2001420.2001442.