FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatik

Improved Integration of Plagiarism Detection in Artemis

Philipp Bauch

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatik

Improved Integration of Plagiarism Detection in Artemis

Verbesserte Integration der Plagiatskontrolle in Artemis

Author:	Philipp Bauch
Supervisor:	Prof. Dr. Bernd Brügge
Advisor:	Dr. Stephan Krusche
Date:	15.02.2021

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 15.02.2021 Philipp Bauch

Abstract

Exercises are an essential part of software engineering courses as students can apply the knowledge taught in class to specific problems. When exercises are graded or even part of an exam, it is crucial to assure students have worked on the assignments independently. Especially for large courses, comparing submissions by hand is too time-consuming, and instructors should use automated tools to detect plagiarism efficiently.

The current integration of plagiarism detection in Artemis requires instructors to use external software to review plagiarism incidents and sort out false positives manually. This process involves managing dozens of files and is error-prone as accidentally confounding student submissions can cause false accusations. Differences in each instructor's workflow might lead to inconsistent assessment results.

This thesis improves the existing integration by processing the plagiarism results directly in Artemis and highlighting similarities between submissions by color. Instructors can confirm or deny a plagiarism incident and leave feedback to students whose submissions were confirmed as plagiarized. Artemis displays a similarity distribution of detected results that helps instructors evaluate the plagiarism found. Instructors can filter out submissions irrelevant for comparison to increase plagiarism detection performance.

Zusammenfassung

Übungen sind ein wesentlicher Bestandteil von Software-Engineering-Kursen, da die Studenten das im Unterricht vermittelte Wissen auf spezifische Probleme anwenden können. Wenn Übungen benotet werden oder sogar Teil einer Prüfung sind, muss sichergestellt werden, dass die Studenten die Aufgaben selbstständig bearbeitet haben. Besonders bei großen Kursen ist der Vergleich von Einreichungen per Hand zu zeitaufwändig, und die Dozenten sollten automatisierte Tools einsetzen, um Plagiate effizient zu erkennen.

Die derzeitige Integration der Plagiatserkennung in Artemis erfordert, dass Dozenten externe Software verwenden, um Plagiatsvorfälle zu überprüfen und falsch erkannte Plagiate manuell auszusortieren. Dieser Prozess umfasst die Verwaltung von Dutzenden von Dateien und ist fehleranfällig, da versehentliche Verwechslungen von Einreichungen zu falschen Anschuldigungen führen können. Unterschiede in den Arbeitsabläufen der einzelnen Dozenten können zu inkonsistenten Bewertungsergebnissen führen.

Die vorliegende Arbeit verbessert die bestehende Integration, indem sie die Plagiatsergebnisse direkt in Artemis verarbeitet und Ähnlichkeiten zwischen Einsendungen farblich hervorgehoben werden. Die Dozenten können einen Plagiatsvorfall bestätigen oder verwerfen und den Studenten Kommentare zu Plagiatsvorwürfen hinterlassen. Artemis zeigt eine Ähnlichkeitsverteilung der gefundenen Ergebnisse an, die den Dozenten hilft, gefundene Plagiate zu bewerten. Dozenten können für die automatisierte Plagiatserkennung irrelevante Einreichungen herausfiltern, um deren Dauer zu reduzieren.

Contents

API Application Programming Interface

CLI Command Line Interface

CSS Cascading Style Sheets

GUI Graphical User Interface

HTML Hypertext Markup Language

REST Representational State Transfer

TMS Text Matching Software

UI User Interface

UML Unified Modeling Language

URPS Usability Reliability Performance Supportability

Chapter 1

Introduction

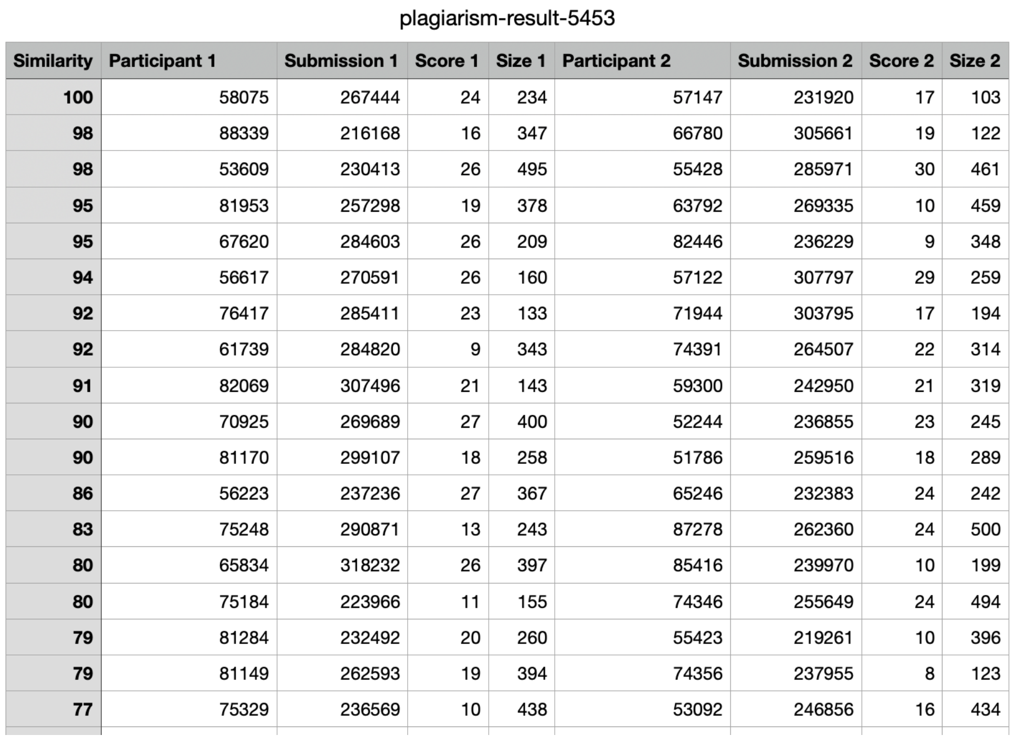
Artemis is an interactive learning platform that offers course instructors the possibility to create and manage online exercises of five different types: *quiz*, *text*, *programming*, *file upload*, and *modeling* [KS18].

Students can then participate in an exercise and upload submissions that will be graded either automatically or manually by the instructors. Part of the assessment is ensuring students solved the given task on their own. For this, instructors can use Artemis to trigger a plagiarism detection algorithm to check submissions for similarity. Each type of exercise might require a different algorithm to detect plagiarism.

1.1 Problem

The COVID-19 pandemic forced most activities to move online, including teaching. Most universities no longer conduct exams on-site but online as well. Among other online education difficulties, many universities reported widespread cheating in online examinations in Spring 2020 [BM21]. To properly conduct both exams and other types of graded assignments, plagiarism detection is an essential part of modern learning platforms.

Artemis provides instructors with a simple integration of automated plagiarism detection tools for modeling, programming, and text exercises. However, relying on automated detection alone runs the risk of students being accused of plagiarism due to a false positive [BCJ^{+}09], even though they solved the problem independently. That's why the automated plagiarism detection results always have to be checked manually by the instructors. However, the current integration is very limited and requires the instructors to perform several manual steps to process the results. They have to use third-party software to manage and review dozens of files generated by the automated plagiarism detection software. This process can be error-prone, and the use of different tools and differences in each instructor's workflow might lead to inconsistent assessments of the results.



plagiarism-result-5453

Similarity	Participant 1	Submission 1	Score 1	Size 1	Participant 2	Submission 2	Score 2	Size 2
100	58075	267444	24	234	57147	231920	17	103
98	88339	216168	16	347	66780	305661	19	122
98	53609	230413	26	495	55428	285971	30	461
95	81953	257298	19	378	63792	269335	10	459
95	67620	284603	26	209	82446	236229	9	348
94	56617	270591	26	160	57122	307797	29	259
92	76417	285411	23	133	71944	303795	17	194
92	61739	284820	9	343	74391	264507	22	314
91	82069	307496	21	143	59300	242950	21	319
90	70925	269689	27	400	52244	236855	23	245
90	81170	299107	18	258	51786	259516	18	289
86	56223	237236	27	367	65246	232383	24	242
83	75248	290871	13	243	87278	262360	24	500
80	65834	318232	26	397	85416	239970	10	199
80	75184	223966	11	155	74346	255649	24	494
79	81284	232492	20	260	55423	219261	10	396
79	81149	262593	19	394	74356	237955	8	123
77	75329	236569	10	438	53092	246856	16	434

Figure 1.1: CSV file containing the results of automatic plagiarism detection sorted by similarity

It is particularly difficult to compare student submissions that are part of a plagiarism incident. Figure 1.1 presents an example file generated by plagiarism detection software, which has a numeric reference to the involved submissions but does not contain their actual content. Therefore, instructors have to manually download and compare the submissions to confirm or dismiss the plagiarism incident.

1.2 Motivation

This thesis aims to provide course instructors with powerful and easy to use tools to detect plagiarism and assess suspected submissions. In the following, we explain our motivation to improve the integration of plagiarism detection in Artemis.

Facilitate online exams. As possibly hundreds of students take part in online exams, manually checking student submissions for plagiarism is not feasible. Automated plagiarism detection can be an effective counter to cheating in online examinations and assuring compliance with academic regulations.

Increase student learning outcomes. Interactive exercises enhance the learning experience of students in online courses [KvFA17]. To maximize the learning outcome, it is essential that students solve problem assignments independently and don't copy from each other. A convenient and robust plagiarism detection system can disincentivize students from cheating.

Reduce administrative work for instructors. As described in previous sections, checking plagiarism detection results can be a time-consuming task for instructors. We want to reduce this effort and give instructors more resources to focus on teaching rather than on administrative tasks.

1.3 Objectives

This thesis defines three central objectives for the improved integration of plagiarism detection in Artemis based on the previous sections.

Decrease complexity of plagiarism detection. The use of automated plagiarism detection has so far led to a considerable administrative burden for course instructors. By improving the integration of plagiarism detection in Artemis, instructors should have an easy way to check the automatically detected results without external software. Also, students should be informed directly in Artemis about plagiarism accusations.

Increase consistency of plagiarism detection results. Because a plagiarism offense can have severe consequences for students, the risk of false accusations must be minimized. Therefore, course instructors must have an efficient way to review the plagiarism results that were automatically detected. An integrated view that displays student submissions side-by-side should further facilitate manual inspection.

Give students more insights into plagiarism accusations. Students should be able to understand better why instructors accused their submission of plagiarism. Giving students direct insight into the submissions involved and providing them with additional feedback from the instructors can help them relate to the allegation and respond appropriately.

1.4 Outline

The outline of this thesis resembles the main software development activities as described in [BD09]. Chapter 2 gives the necessary theoretical background to understand the topic of this thesis. Chapter 3 identifies the proposed system's requirements, refers them to the current system, and presents the analysis models. Chapter 4 addresses the requirements by specifying design goals and transforms the analysis model into a system design model. Chapter 5 specifies selected objects of the solution domain. Chapter 6 summarizes the results of this thesis and gives an outlook on potential future work.

Chapter 2

Background

Plagiarism is defined as the practice of taking someone else's work or ideas and passing them off as one's own [Dic89]. The increasing usage of computers and the internet in online education have made it easier to plagiarize the work of others [CL01]. In the academic world, most institutions consider plagiarism by students a severe offense that is subject to sanctions such as a failing grade on the particular assignment, the entire course, or even being expelled from the institution [Smi18]. According to [Tur], the three most common forms of plagiarism committed by students are:

- Submitting someone's work as their own.
- Taking passages from their previous work without adding citations (self-plagiarism).
- Re-writing someone's work without properly citing sources.

Comparing student submissions by hand is the most traditional form of plagiarism detection but can be a time-consuming task [BM09]. Section 2.1 explains the fundamentals of computer-assisted plagiarism detection and presents some examples. In Section 2.2, we discuss common types of errors in binary classification problems like plagiarism detection.

2.1 Computer-assisted plagiarism detection

Nowadays, course instructors can use text-matching software (TMS) to find similar passages of text in a set of student submissions. TMS saves academic staff much time by quickly matching student assignments with electronic sources. Most TMS programs match assignments submitted by students with other students' submissions and an electronic archive of articles and web documents [KS07].

The following list briefly describes three common techniques of TMS to detect plagiarism in text documents:

- **Fingerprinting:** This method tries to find a compact representation of the content, also referred to as *fingerprint*, of each given document. For this, key substrings of a document are mapped to an integer using a mathematical function. TMS compares the resulting set of numbers with other fingerprints and determines a score representing the number of elements two fingerprints have in common [HZ03].

- **String matching:** The goal of string matching is to find exact duplications in a set of strings by detecting matches. Matches are pairs of identical substrings that cannot be extended to the left or right. String matching algorithms commonly use the suffix tree data structure to efficiently query substrings of documents [Bak93].

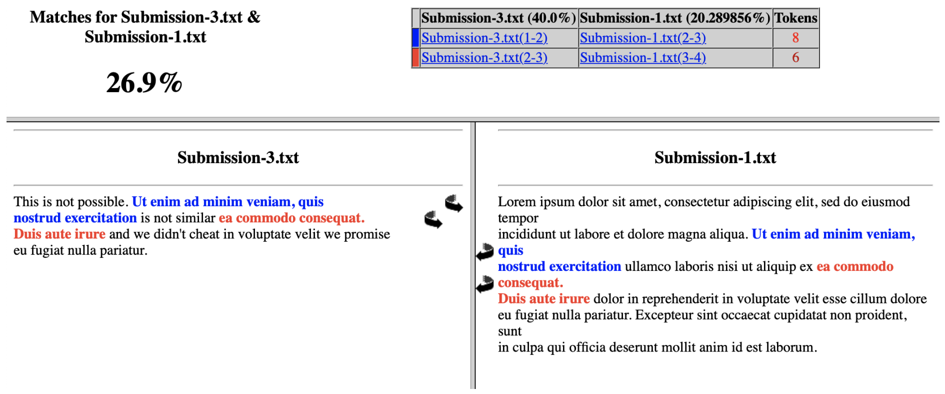
- **Stylometry:** This approach attempts to detect stylistic changes in a document by extracting features from the text that quantify aspects of an author's writing style [BRC19]. Passages that differ stylistically from others in the same document are marked as potentially plagiarized.

Detecting plagiarism in computer programs requires different methods compared to plagiarism detection in text documents. In most cases, students disguise copied code intentionally to make plagiarism more challenging to detect: they rename variables, add whitespace and comments, or restructure code while keeping programs functionally equivalent. Traditional TMS don't work well in those cases, and therefore, source code submissions must be normalized, for example, by using a lexer to convert a program into a list of tokens [RC07].

2.1.1 JPlag

JPlag is a system for detecting plagiarism among a set of programs [PMP00] and is currently used in the Artemis system. While JPlag also works for text documents, it initially supported plagiarism detection for source code written in Java, Scheme, C, and C++ [PMP00].

First, JPlag uses a parser front-end to normalize documents and transform source code into a string of tokens. Second, it implements the Greedy String Tiling algorithm to compare pairs of token strings and determine their similarity. Greedy String Tiling can deal with the transposition of substrings and identify the longest duplicate text sequence between two documents [Wis93]. In JPlag, instructors can specify the minimum length of duplicate fragments to avoid the detection of irrelevant similarities. After the pairwise comparison of documents is complete, JPlag generates web pages of the results highlighting similarities between two suspected submissions, as seen in Figure 2.1.


Matches for Submission-3.txt & Submission-1.txt

26.9%

	Submission-3.txt (40.0%)	Submission-1.txt (20.289856%)	Tokens
🟦	Submission-3.txt(1-2)	Submission-1.txt(2-3)	8
🟥	Submission-3.txt(2-3)	Submission-1.txt(3-4)	6

Submission-3.txt

This is not possible. **Ut enim ad minim veniam, quis nostrud exercitation** is not similar **ea commodo consequat. Duis aute irure** and we didn't cheat in voluptate velit we promise eu fugiat nulla pariatur.

Submission-1.txt

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. **Ut enim ad minim veniam, quis**
nostrud exercitation ullamco laboris nisi ut aliquip ex **ea commodo consequat.**
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt
in culpa qui officia deserunt mollit anim id est laborum.

Figure 2.1: JPlag report of two similar student submissions

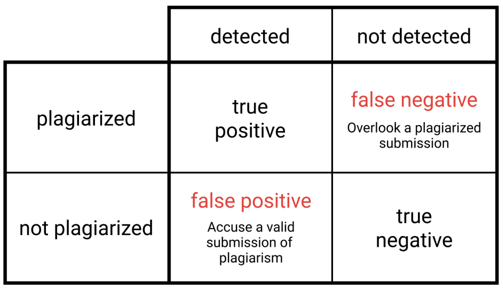
Initially provided as a web service, JPlag is only available as a command-line interface (CLI) application and is thus difficult to integrate into external software systems. We cover more details about developing an application programming interface (API) for JPlag and integrating it in Artemis in Chapter 5.

2.1.2 Moss

Moss (for **M**easure **O**f **S**oftware **S**imilarity) is another popular tool for detecting duplication within a set of source code documents. It was released in 1997 and is widely used in engineering courses to detect plagiarism in programming assignments [Aik20].

Moss is provided as an internet service. Its usage involves packaging up students' submissions, uploading them for automated comparison, and inspecting the results. The server responds with generated web pages that highlight similarities between student submissions similar to JPlag. Instructors can also provide template code that student submissions are based on to eliminate false positives that arise from legitimate sharing of code [Aik20].

Compared to JPlag, which applies a string matching technique on a set of normalized tokens, Moss implements a copy-detection algorithm based on fingerprinting of the documents after whitespace, identifiers, and specific language keywords are removed. Moss uses the winnowing algorithm to select the fingerprints to be compared for each document [SWA03], but it is out of the scope of this thesis to explain the algorithm in more detail.



	detected	not detected
plagiarized	true positive	false negative Overlook a plagiarized submission
not plagiarized	false positive Accuse a valid submission of plagiarism	true negative

Figure 2.2: Confusion matrix of binary classiciation for plagiarism detection

2.2 Errors in binary classification

In statistics, classification is the task of determining to which of a set of categories a new observation belongs [MST95]. Detecting plagiarism can be considered a binary classification problem. Given a set of student submissions, the task is to split the set into two groups, of which one contains plagiarized submissions. Figure 2.2 shows a confusion matrix with four different combinations of predicted and actual values. Confusion matrices can be used to visualize a classification algorithm's performance and to calculate the relative proportion of different types of errors on the total set of observations [V.S97].

In the context of plagiarism detection, a type I error (false positive) occurs if the system accuses a submission of plagiarism that the student solved independently; a type II error (false negative) occurs if a plagiarized submission is undetected. It's difficult to avoid type I and type II errors entirely [BCJ$^+$09]. Reducing one type of error generally results in increasing the other type of error [SB75]. Hence, we trade off error rates against each other: if we want to detect all plagiarism incidents, we have to accept false accusations; if we don't want to make false accusations, we won't catch all plagiarized submissions.

As discussed previously, plagiarism offense is subject to sanctions that can have severe consequences for students. Therefore, reducing the risk of false accusations while detecting as many plagiarized submissions as possible is a fundamental design goal of the proposed system and will be addressed in more detail in Chapter 4.

Chapter 3

Requirements Analysis

This chapter covers two of the main object-oriented software engineering activities: requirement elicitation and analysis [BD09]. Section 3.1 gives an overview of the proposed system's purpose, scope, objective, and success criteria. Section 3.2 describes the current system and depicts the problems faced by it. Following this, Section 3.3 describes the changes proposed by this thesis and lists the functional and non-functional requirements for the proposed system. Lastly, Section 3.4 models the application domain in the form of an analysis model in the semi-formal Unified Modeling Language (UML) [BD09].

3.1 Overview

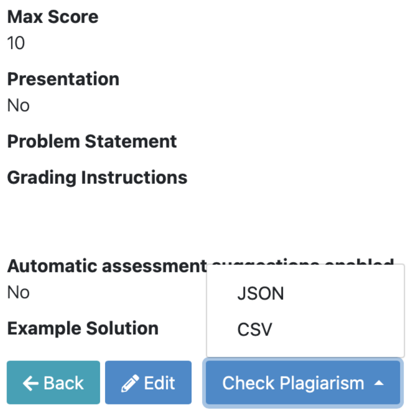
Purpose: The purpose of the proposed system is to facilitate manual inspection of plagiarism results. Instructors should not need to rely on external software to manage plagiarism incidents and compare suspected submissions. The proposed system should also support communication between students and instructors on plagiarism accusations.

Scope: We will concentrate on improving the integration of plagiarism detection software used in the current system. Adding new tools or implementing alternative algorithms is beyond the scope of this thesis.

Objectives: The proposed system's primary goal is to provide a user interface (UI) to compare similar submissions side by side. Instructors should be able to confirm or dismiss plagiarism incidents to sort out false positives. Automatic plagiarism detection should be configurable, and students should get notified about plagiarism accusations on their submissions.

Success Criteria: We infer the three main success criteria from the purpose of the proposed system described above:

- Artemis supports side-by-side comparison of suspected submissions and highlights similarities
- Instructors can confirm or dismiss plagiarism incidents to sort out false positives
- Students get notified about plagiarism accusations and receive feedback directly in Artemis



Max Score
10

Presentation
No

Problem Statement

Grading Instructions

Automatic assessment suggestions enabled
No

Example Solution

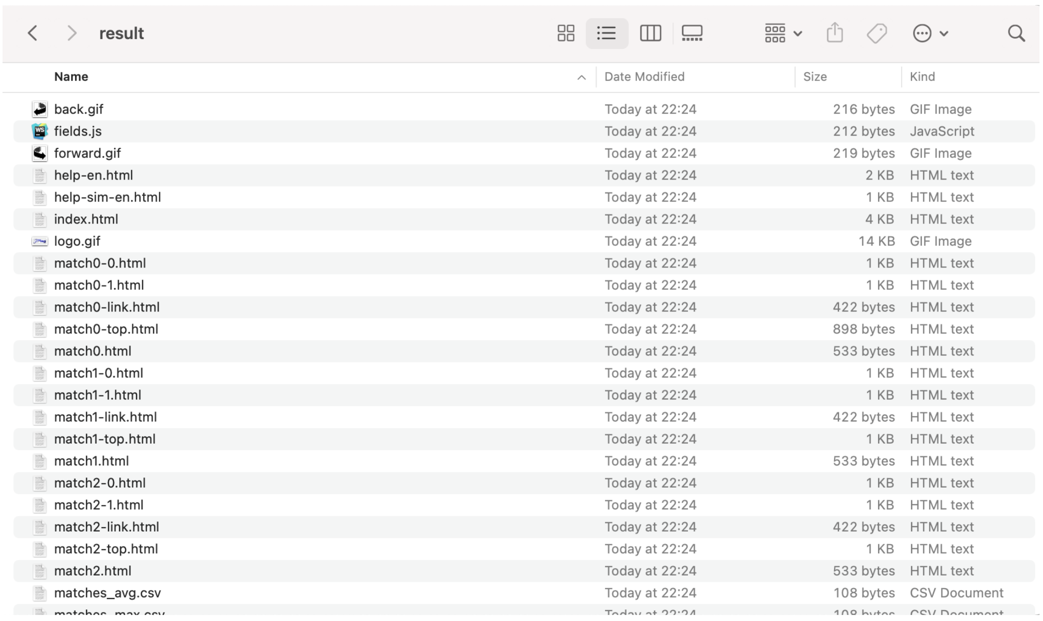
Figure 3.1: The current system allows instructors to download the plagiarism results in JSON or CSV format

3.2 Current System

Automatic plagiarism detection is available for modeling, text, and programming exercises. Instructors can download the results generated by plagiarism detection software after the automated comparison is complete. It is not possible to view the results directly in Artemis. Instructors have to manually open two separate browser windows to compare suspected submissions and sort out false positives. Artemis does not highlight similarities between modeling exercise submissions, making them more difficult to compare. Most instructors use Microsoft Excel[1] to manage the status of plagiarism incidents

[1] https://www.microsoft.com/en-us/microsoft-365/excel

and keep track of student responses to plagiarism accusations. This process puts a lot of administrative burden on the instructors, and accidentally confounding student submissions can lead to an improper allegation of plagiarism.


result

Name	Date Modified	Size	Kind
back.gif	Today at 22:24	216 bytes	GIF Image
fields.js	Today at 22:24	212 bytes	JavaScript
forward.gif	Today at 22:24	219 bytes	GIF Image
help-en.html	Today at 22:24	2 KB	HTML text
help-sim-en.html	Today at 22:24	1 KB	HTML text
index.html	Today at 22:24	4 KB	HTML text
logo.gif	Today at 22:24	14 KB	GIF Image
match0-0.html	Today at 22:24	1 KB	HTML text
match0-1.html	Today at 22:24	1 KB	HTML text
match0-link.html	Today at 22:24	422 bytes	HTML text
match0-top.html	Today at 22:24	898 bytes	HTML text
match0.html	Today at 22:24	533 bytes	HTML text
match1-0.html	Today at 22:24	1 KB	HTML text
match1-1.html	Today at 22:24	1 KB	HTML text
match1-link.html	Today at 22:24	422 bytes	HTML text
match1-top.html	Today at 22:24	1 KB	HTML text
match1.html	Today at 22:24	533 bytes	HTML text
match2-0.html	Today at 22:24	1 KB	HTML text
match2-1.html	Today at 22:24	1 KB	HTML text
match2-link.html	Today at 22:24	422 bytes	HTML text
match2-top.html	Today at 22:24	1 KB	HTML text
match2.html	Today at 22:24	533 bytes	HTML text
matches_avg.csv	Today at 22:24	108 bytes	CSV Document
matches_max.csv	Today at 22:24	108 bytes	CSV Document

Figure 3.2: Result files generated by plagiarism detection software used in the current system

For courses with many students, a pairwise comparison of submissions for plagiarism detection can take a long time. Regardless of course size and instructor preferences, automatic plagiarism detection is not configurable. For example, instructors have no way to specify which submissions to ignore for plagiarism detection. Skipping low-scoring submissions because they are of low relevance to plagiarism could significantly increase automated plagiarism detection performance. Also, the plagiarism detection results are not stored in the current system and get lost after downloading. If multiple instructors want to access the results, they must either run the plagiarism detection themselves or share the generated results via other means.

3.3 Proposed System

The proposed system builds on the existing integration for plagiarism detection to make the manual inspection of plagiarism incidents more efficient and less error-prone. It introduces an integrated workflow for instructors to compare and manage similar student submissions.

In particular, the proposed system adds a new page for instructors to configure the automatic plagiarism detection, compare detected submissions side by side, and keep track of plagiarism incidents' status. Additionally, the proposed system highlights similarities between submissions by color and gives instructors access to statistics about a plagiarism detection run. Detected matches get sorted by similarity, and instructors can write comments on confirmed plagiarism incidents to give students detailed feedback.

3.3.1 Functional Requirements

This section describes the functional requirements (FR) of the proposed system. Functional requirements define what the proposed system should do and describe the interactions between the system and its environment independent of its implementation [BD09]. The client and the user evaluate the listed requirements using the six validation criteria: correctness, clarity, completeness, consistency, realism, and traceability as described in [BD09].

FR1 **Compare submissions side by side**: Instructors can display similar submissions side by side to facilitate manual inspection.

FR2 **Manage detected plagiarism matches**: Instructors can confirm or deny the results generated by the automatic plagiarism detection to sort out false positives.

FR3 **View plagiarism statistics**: The system presents a similarity distribution of all reported incidents to the instructors to help them evaluate the detected results.

FR4 **Highlight similarities**: Similarities between submissions are highlighted by color to make the inspection more efficient.

FR5 **Configure automatic plagiarism detection**: Instructors can filter submissions for plagiarism detection by specifying a minimum score and size. They can also specify the minimum similarity two submissions must have to get reported.

FR6 **Add detailed feedback**: Instructors can provide additional feedback for the students on why they confirmed a plagiarism incident.

FR7 **Respond to feedback**: Students can respond to the plagiarism accusation directly in Artemis and justify the validity of their submission.

FR8 **See plagiarism detection progress**: Instructors see information about the number of pending comparisons and the estimated time until completion.

FR9 **Learn from manual inspection**: The system improves automatic plagiarism detection over time by learning from the instructors' manual inspection results.

FR10 **Detect group plagiarism**: The system detects not only pairs, but also groups of two or more similar submissions.

3.3.2 Nonfunctional Requirements

In this section, we list the nonfunctional requirements (NFR) of the proposed system. In particular, we categorize all quality requirements according to the Usability Reliability Performance Supportability (URPS) model described in [BD09].

Usability

NFR1 **Ease of Use**: Instructors can start, configure, and inspect the results of automated plagiarism detection from one page.

NFR2 **Responsiveness**: Side-by-side comparison of submissions should work on desktop and tablet screens.

NFR3 **Efficiency**: Loading submissions and updating a plagiarism incident's status can be done by at most three clicks.

Reliability

NFR4 **Consistency**: Automated plagiarism detection should generate identical results if submissions and configuration stay unchanged. The integration should enforce a consistent judgement on detected plagiarism and reduce the number of complaints by students.

NFR5 **Robustness**: Students must be informed if their submission has been accused of plagiarism within one day. It should always be possible for students to respond to the accusation.

Performance

NFR6 **Response Time**: Automatic plagiarism detection should be feasible even for large courses and take no longer than two minutes per 100 submissions.

NFR7 **Load Time**: The manual inspection of the results in the browser should be possible at 30 frames per second even for large courses with possibly hundreds of plagiarism incidents.

Supportability

NFR8 **Extensibility**: The system should be extensible and allow to integrate additional plagiarism detection software.

3.4 System Models

This section describes the system models, which are formed from the requirement analysis.

3.4.1 Scenarios

Scenarios are a technique to describe the requirements and bridge the conceptual gap between end-users and developers. They explain the system's use as a series of interactions from the viewpoint of a single actor [BD09]. We divide the scenarios described in this section into visionary and demo scenarios. Visionary scenarios describe a potential future system that would entirely solve the problem at hand, even if it is not realizable. In contrast, demo scenarios describe the proposed system, including the previous section's functional requirements.

Visionary Scenario: Inspect detected plagiarism matches in real time

Alice teaches a software engineering course that uses Artemis to manage programming exercises. It is important to Alice that students solve their assignments independently. Therefore, Alice runs the automated plagiarism detection in Artemis, which checks student submissions for similarities. Although many students participate in Alice's course, she doesn't have to wait long for the results because detected plagiarism is returned in real-time as soon as any are detected. Alice also receives precise information about the progress of plagiarism detection and the number of pending comparisons.

The system learned from Alice's previous decisions when sorting out false positives and improved its algorithm. This time, she can confirm all detected plagiarism incidents. Alice saves time during manual inspection since clicking on a detected match in one submission leads her directly to the other submission's corresponding match. This feature comes in handy when student submissions span multiple files, and manually navigating to similar sections is time-consuming.

Demo Scenario: Automatic plagiarism detection for large courses

Bob is the instructor of a university course with several hundred students. As part of a graded homework assignment, Bob has created a programming exercise in Artemis with a maximum score of 10 points. The due date has already passed, and Bob and his teaching assistants have graded all the submissions. To make sure that students have not copied from each other while working on the assignments, Bob starts the automated plagiarism detection. Bob knows that for courses with many students, the system must perform numerous comparisons. To reduce the time required for plagiarism detection, Bob adjusts some configuration options. Because submissions that have received a low score are irrelevant for plagiarism detection, the software should only consider submissions with a score of at least 40%. Also, Bob configures the automated plagiarism detection to return only comparisons with a similarity of at least 75%. After the automated plagiarism check is completed, Bob checks the results manually. Bob can confirm suspected plagiarism in two cases and writes detailed feedback for the involved students on why he considers their submissions invalid. His students can communicate with Bob directly in Artemis to respond to the plagiarism accusation.

3.4.2 Use Case Model

The use case model describes the functional behavior of the system as seen by the user [BD09]. This functional model later delivers new methods for the object model. The actors interacting with the proposed system as an external entity are:

- **Instructor:** Starts the plagiarism detection and inspects the results.
- **Student:** Reviews and responds to a plagiarism accusation.

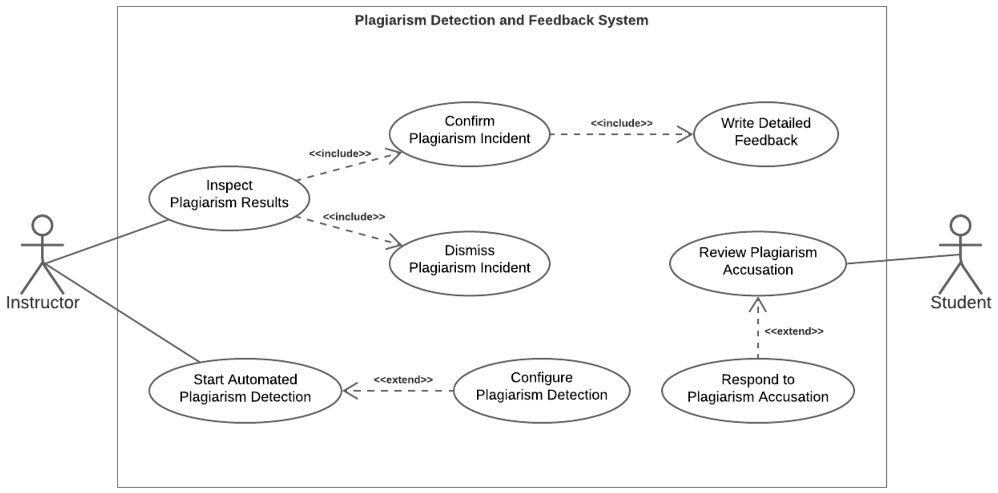

Plagiarism Detection and Feedback System
Instructor
Student
Inspect Plagiarism Results
Confirm Plagiarism Incident
Write Detailed Feedback
Dismiss Plagiarism Incident
Review Plagiarism Accusation
Start Automated Plagiarism Detection
Configure Plagiarism Detection
Respond to Plagiarism Accusation
<<include>>
<<include>>
<<include>>
<<extend>>
<<extend>>

Figure 3.3: Use case diagram of the plagiarism detection and feedback system in Artemis

Base Use Cases

Start the automated plagiarism detection: Instructors can trigger the automated plagiarism detection once the exercise's due data has passed and all student submissions were uploaded. The system then loads all submissions and performs a pairwise comparison.

Inspect the plagiarism detection result: After the automated plagiarism detection is completed, the system returns all detected submissions to the instructor. To avert the danger of false plagiarism accusations, the instructor manually reviews the results and sorts out false positives.

Review a plagiarism accusation: Students have the possibility to review the plagiarism incident if it was confirmed by an instructor. This involves viewing their submission next to the other submission involved to see similarities, as well as reading the feedback provided by the instructor.

Extension Use Cases

Configure automated plagiarism detection: To reduce the number of submissions to compare and thus decrease the time needed for plagiarism detection, instructors can set options to sort out submissions irrelevant for plagiarism detection. For example, they can specify a minimum score in percent a submission must have in order to be compared.

Respond to the plagiarism accusation: Students might want to respond to the plagiarism accusation and justify their submissions validity. The feedback loop between instructor and student can be repeated multiple times.

Inclusion Use Cases

Confirm a plagiarism incident: When checking the automatically reported plagiarism incident, the instructor can confirm the accusation. This will mark the involved submissions as invalid so that they are excluded from the exercise. It is up to the instructor to further punish the involved students.

Dismiss a plagiarism incident: It is also possible for an instructor to dismiss detected plagiarism if it is considered a false positive. In this case, the instructor resolves the incident and no further steps will be taken.

Write detailed feedback: If a plagiarism incident was confirmed, the instructor has the option to provide the students with additional information on why their submission was accused of plagiarism. This can help students to properly respond to the accusation and clarify misunderstandings.

3.4.3 Analysis Object Model

This section describes the application domain in terms of the static structure of the proposed system. Figure 3.4 models the plagiarism detection and feedback system with an analysis object model illustrating the relevant objects, their attributes and methods. Access modifiers or methods' return types are omitted for abstraction.

Exercise: Students can participate in an exercise and create submissions with their solution. After the exercise's *dueDate*, instructors can detect plagiarism among all students submissions.

Submission: Submissions represent a student's solution to the problem statement of a given exercise. If a submission is too similar to

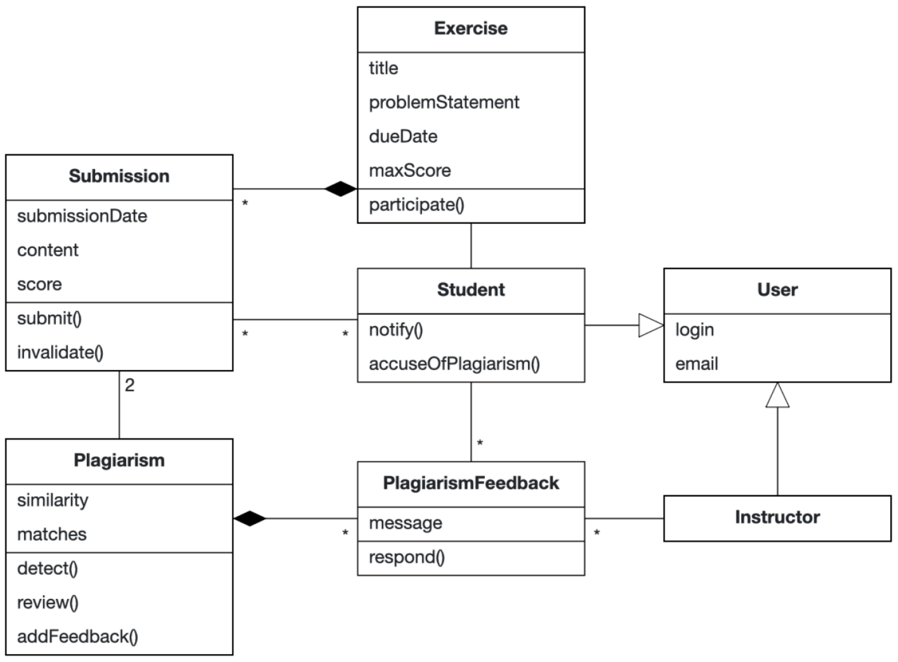


Figure 3.4: Analysis object model of the plagiarism detection and feedback system

another student's submission, instructors can accuse it of plagiarism. Once the plagiarism incident has been reviewed and confirmed, the submission is marked invalid.

Plagiarism: A plagiarism object relates to a pair of two similar submissions. It contains a list of *matches* containing elements present in both submissions. Its *similarity* property indicates to which degree both submissions equal. Additionally, instructors can review a plagiarism incident and add feedback for the student, if the plagiarism was confirmed. Students can only review plagiarism objects they are directly involved in.

Plagiarism Feedback: Instructors can add feedback to an incident to give students more insights on why their submission was accused of plagiarism. Students can respond to the feedback to justify their submission's validity.

Student: If a student's submissions was marked as plagiarism, instructors notify the student about the incident.

3.4.4 Dynamic Model

This section describes the proposed system's internal behavior with an activity diagram, which visualizes the control and data flow of plagiarism detection. The flow of events in the proposed system does not differ much from the activities related to plagiarism detection in the current system. However, the proposed system aims at integrating the presented workflow for plagiarism detection more directly into Artemis. We will cover details about the integration in Chapter 4.

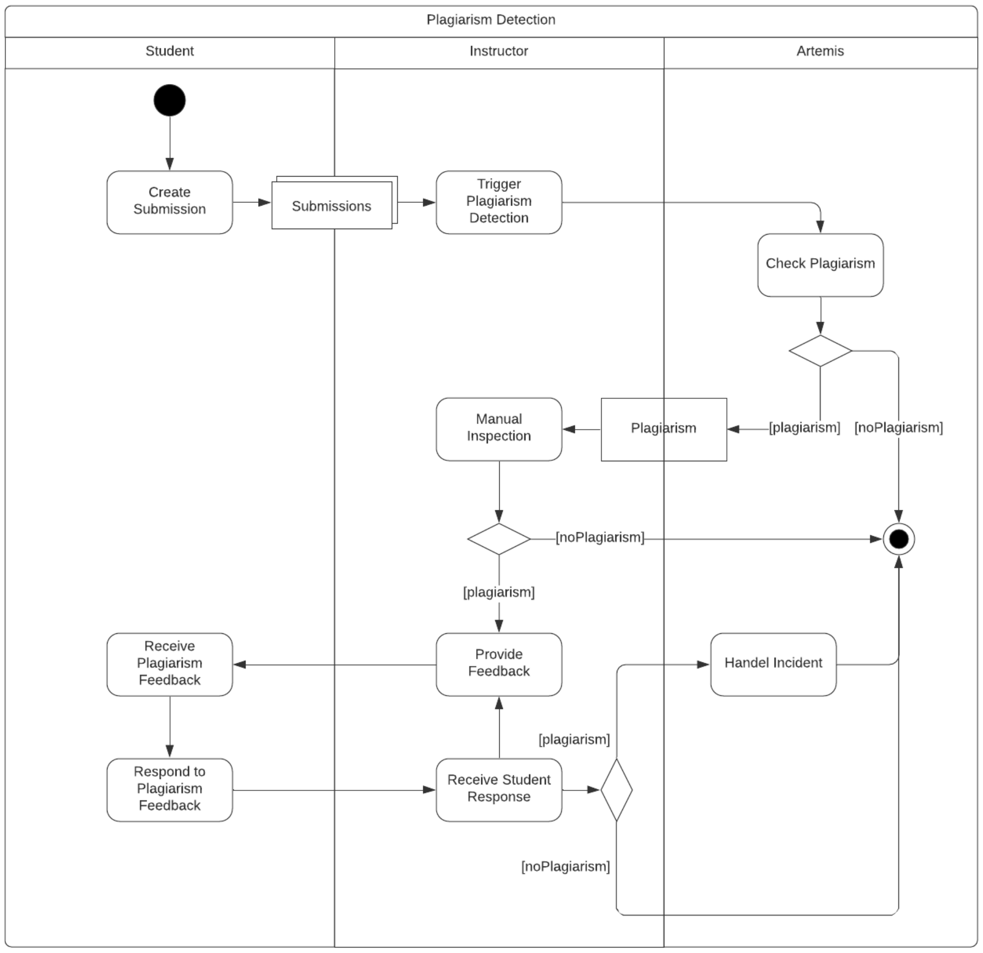


Figure 3.5: Activity diagram visualizing the flow of events of plagiarism detection and the feedback loop in Artemis.

Figure 3.5 groups the presented activities in three swimlanes, each representing a different actor in the plagiarism detection and feedback system. Although courses on Artemis usually have many students participating in an exercise, the left swimlane only describes a single student's activities. As part of the plagiarism detection flow, students first participate in an exercise and create submissions. In the next step, the instructor triggers the automatic plagiarism detection to compare the created submissions with each other. For each pair of student submissions, the Artemis system automatically checks whether the given submissions exceed a given similarity threshold. If their similarity is insignificant, the system will ignore the pair of submissions. Otherwise, the Artemis system passes a new plagiarism object to the instructor for manual inspection.

From this point on in the control and data flow presented in Figure 3.5, only a single plagiarism incident will be considered at once to demonstrate the workflow in a specific case. Manual inspection allows the course instructor to dismiss any plagiarism incident that may have been wrongly detected by the Artemis system, thus averting the danger of false accusations. If the instructor confirms the plagiarism incident, he provides additional feedback for the students involved. The student receives the plagiarism accusation together with the instructor's feedback. In response, the student can justify his submission's validity, to which the instructor can respond again. It is possible to repeat this feedback loop multiple times. In the last step, given the student's feedback, the instructor can dismiss the plagiarism incident again. In total, a pair of submissions must pass three checks to be accused of plagiarism: automatic detection, manual inspection, and instructor's judgment after the feedback loop. This workflow reduces the likelihood of false accusations.

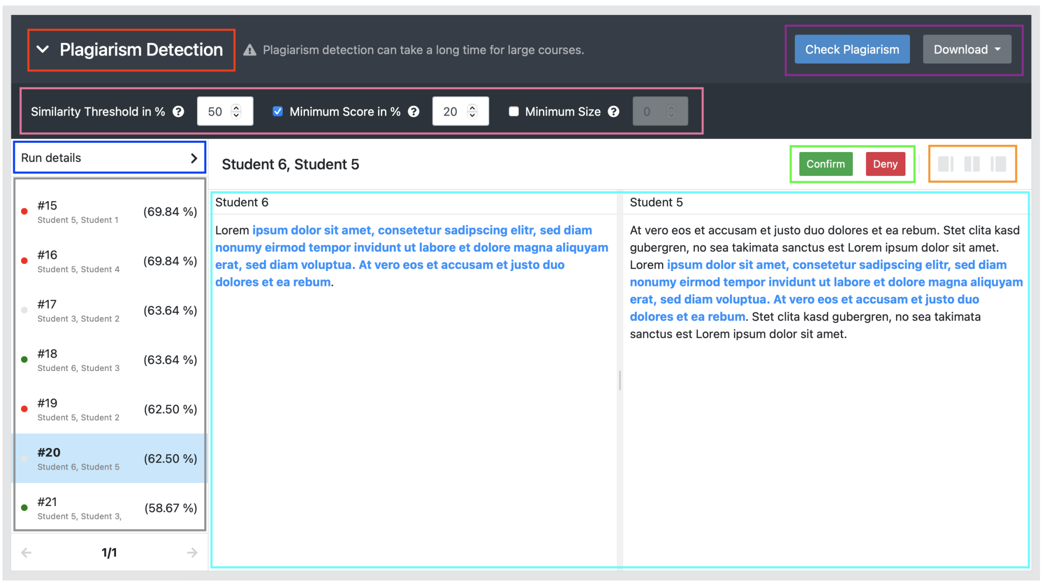
3.4.5 User Interface

This section depicts and explains the rational of the plagiarism detection UI in the proposed system. Figure 3.6 shows the entire plagiarism detection page with two similar submission displayed side by side. Figure 3.8 shows additional information about the detected results and the detection run itself.

Plagiarism Split View

Instructors can start and configure plagiarism detection from the page header at the top. The page also has a sidebar to display a list of detected plagiarism. However, the main area of the page is reserved for the split view that helps instructors compare student submissions. A detailed explanation of the UI components is listed below.

Red: By clicking on the page header, instructors can expand or collapse the configuration panel. After automatic plagiarism detection



Plagiarism detection can take a long time for large courses.

Similarity Threshold in % 50

[✓] Minimum Score in % 20

[ ] Minimum Size 0

#15 Student 5, Student 1	(69.84 %)
#16 Student 5, Student 4	(69.84 %)
#17 Student 3, Student 2	(63.64 %)
#18 Student 6, Student 3	(63.64 %)
#19 Student 5, Student 2	(62.50 %)
#20 Student 6, Student 5	(62.50 %)
#21 Student 5, Student 3,	(58.67 %)

Student 6, Student 5

Student 6

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.

Student 5

At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Figure 3.6: Plagiarism detection page of the proposed system showing the split view that highlights similarities between two suspected submissions

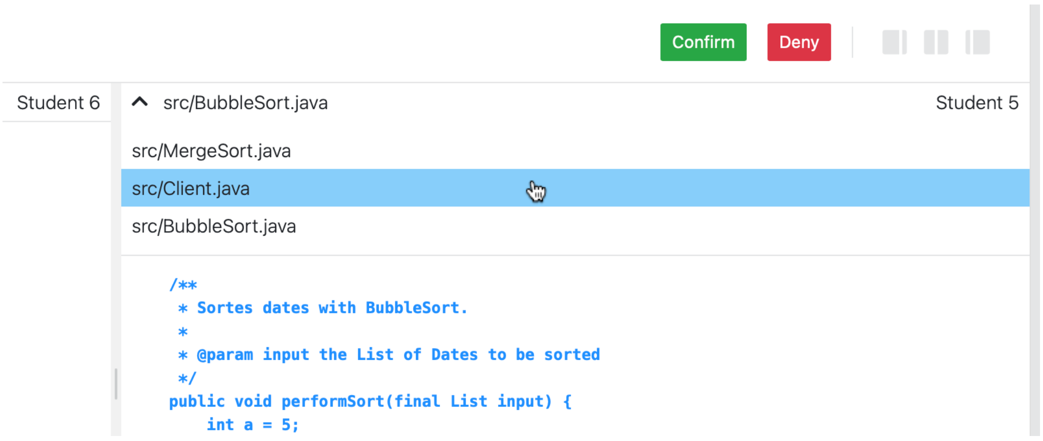
finishes, more vertical space might be useful to compare the detected submissions more easily.

Pink: The configuration panel contains multiple options instructors can adjust to their preferences. Instructors must enable optional configuration values like *Minimum Score in %* and *Minimum Size* by clicking on the related checkbox to take effect.

Purple: The *Check Plagiarism* button triggers a new automated plagiarism detection run with the selected configuration values. Once the results are available, instructors can use the *Download* button to download the results.

Gray: In the side panel on the left, instructors can access all detected plagiarism incidents sorted by similarity. Each item in the list has a small indicator for its status and contains the student logins involved in the plagiarism. Selecting an item from the list displays the involved submissions in the split view.

Cyan: The split view displays two similar submissions side by side. For text and modeling exercises, the split pane header shows the student's login who created the submission. For programming exercises, the split pane header also displays the currently selected file and can be clicked to open the list of files the submission consists of (see Figure 3.7). The system highlights similarities between both submissions by color to make the submissions easier to compare.



```
/**
 * Sortes dates with BubbleSort.
 *
 * @param input the List of Dates to be sorted
 */
public void performSort(final List input) {
    int a = 5;
```

Figure 3.7: Split pane header displaying a list of all files a programming submission consists of

Orange: With the split view controls, instructors can quickly maximize each submission or reset the split view. Although instructors can adjust the split panels by dragging the gutter separating them, these shortcuts can save some time when focusing on the submissions' content.

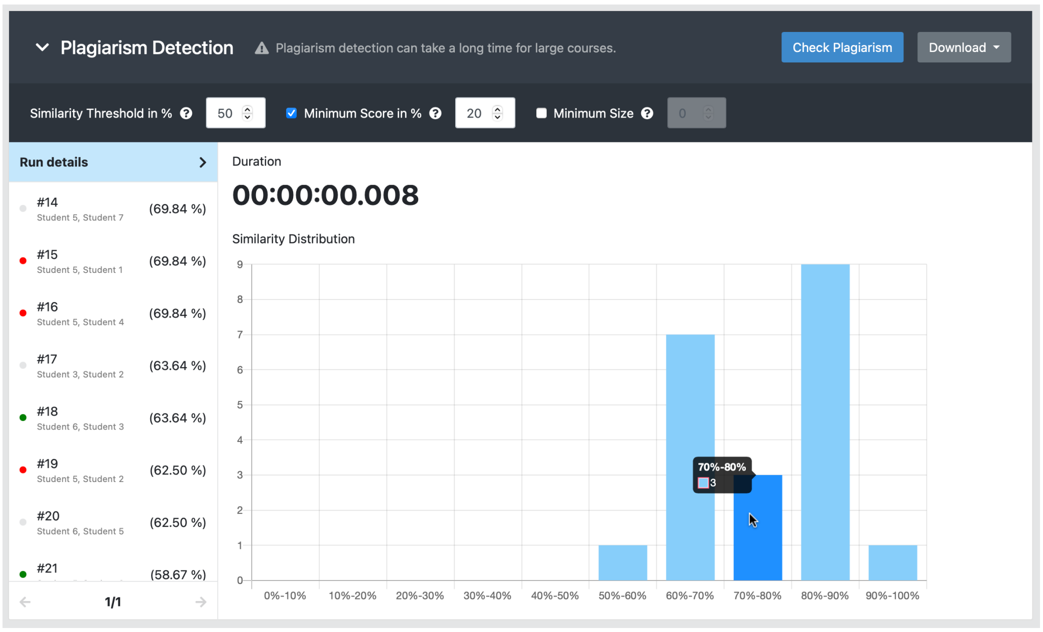
Green: The *Confirm* and *Deny* button update the status of the selected plagiarism incident. This makes it easy for instructors to sort out plagiarism that was incorrectly detected and proceed to manage only confirmed incidents. If a plagiarism was confirmed, instructors can write additional comments for the students, as depicted in Figure 3.9.

Blue: The *Run details* tab at the top of the side panel opens the plagiarism run details view explained in the next section.

Plagiarism Run Details

Figure 3.8 shows the details of an example plagiarism detection run. The view is used to provide instructors with additional information about the run and its results.

The similarity distribution is visualized as a bar chart and indicates how detected plagiarism incidents are distributed in similarity. For example, in short, highly constrained problems, there's not much room for variation and


Plagiarism Detection ⚠ Plagiarism detection can take a long time for large courses.

Similarity Threshold in % 50 | ✓ Minimum Score in % 20 | Minimum Size 0

Run details

#14	Student 5, Student 7	(69.84 %)
#15	Student 5, Student 1	(69.84 %)
#16	Student 5, Student 4	(69.84 %)
#17	Student 3, Student 2	(63.64 %)
#18	Student 6, Student 3	(63.64 %)
#19	Student 5, Student 2	(62.50 %)
#20	Student 6, Student 5	(62.50 %)
#21		(58.67 %)

1/1

Duration

00:00:00.008

Similarity Distribution

Figure 3.8: Run details tab showing the duration of the plagiarism detection run and the similarity distribution of its results

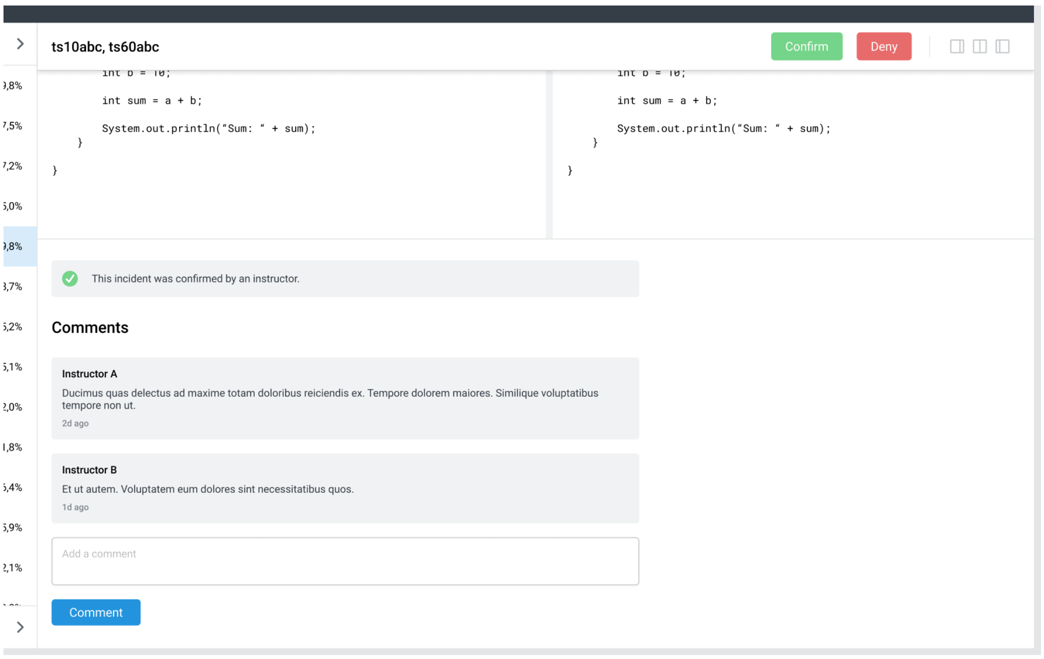
the exercise might yield more similar student submissions due to the limited set of correct solutions. The similarity distribution can help instructors evaluate the detected results.

Plagiarism Comment Section

The comment section is displayed below the split view of a given plagiarism incident. As shown in Figure 3.9, instructors can leave additional comments to provide students with feedback on why they accused their submission of plagiarism. Students also have acccess to the comment section to respond to the received feedback.

Exercise Overview

If instructors accuse a student's submission of plagiarism, a hint for the student is displayed on the respective exercise in the exercise overview. As depicted in Figure 3.10, the student can click on *View incident* to open the plagiarism detection page and see both the involved submissions in the split view and the feedback provided by instructors.



ts10abc, ts60abc

```
        int b = 10;

        int sum = a + b;

        System.out.println("Sum: " + sum);
    }
}
```

```
        int b = 10;

        int sum = a + b;

        System.out.println("Sum: " + sum);
    }
}
```

This incident was confirmed by an instructor.

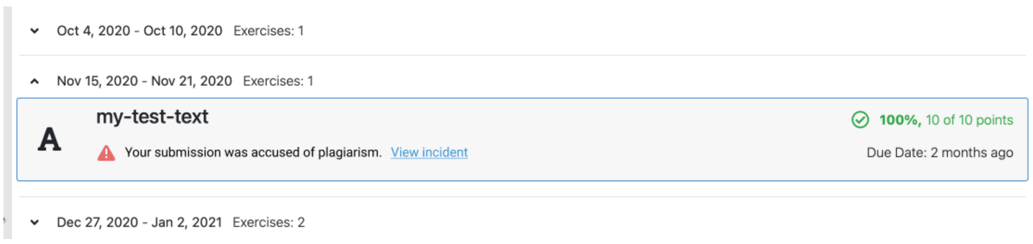
Comments

Instructor A
Ducimus quas delectus ad maxime totam doloribus reiciendis ex. Tempore dolorem maiores. Similique voluptatibus tempore non ut.
2d ago

Instructor B
Et ut autem. Voluptatem eum dolores sint necessitatibus quos.
1d ago

Add a comment

Figure 3.9: Comment section of a plagiarism incident showing the feedback of two instructors



Oct 4, 2020 - Oct 10, 2020 Exercises: 1

Nov 15, 2020 - Nov 21, 2020 Exercises: 1

A my-test-text
Your submission was accused of plagiarism. View incident
100%, 10 of 10 points
Due Date: 2 months ago

Dec 27, 2020 - Jan 2, 2021 Exercises: 2

Figure 3.10: Exercise overview displaying a hint that the student's submission was accused of plagiarism

Chapter 4

System Design

This chapter transforms the analysis model developed in the previous chapter into a system design model by following the system design template in [BD09]. This process aims to bridge the gap between the application domain and the solution domain and design the internal structure of the proposed system in more detail.

4.1 Overview

The proposed system extends Artemis and builds upon its current client-server architecture. The application client provides the graphical user interface (GUI) and handles all interactions with the end-user. It is implemented in HTML, SCSS, and TypeScript and uses the JavaScript-Framework Angular 11 and the CSS-Framework Bootstrap. The application server is written in Java and utilizes the Spring framework and Hibernate. The server offers its business logic through a REST-based Application Programming Interface (API) and WebSockets. Furthermore, it is connected to a MySQL database to store and manipulate the application data. The server also communicates with external systems like the version control (VC) server and the continuous integration (CI) server.

4.2 Design Goals

We derive the design goals presented in this section from the non-functional requirements listed in Section 3.3.2. Design goals describe the system's qualities and values against which alternative designs should be evaluated [BD09]. As non-functional requirements may conflict with each other, we identify trade-offs and order the design goals according to their priority from high to low.

Reliability Courses on Artemis typically have hundreds of students. It is therefore unfeasible for instructors to manually check student submissions for plagiarism. This is why automated plagiarism detection should always be available for instructors as a first step to detect similar student submissions. Although instructors must double-check detected matches, achieving high reliability and consistency of the results is the most important design goal.

Usability To help instructors compare detected submissions efficiently, the UI must provide easy ways to navigate through the results and offer enough screen area to compare detected submissions side by side. The page must also include all necessary controls to manage a plagiarism incident and to write comments optionally.

Performance The automated plagiarism detection performs a pairwise comparison of all student submissions for a given exercise. Besides starting the plagiarism detection, instructors only have to wait for the comparisons to complete to inspect the result. However, for large courses, automatic plagiarism detection has to perform many comparisons. As a result, instructors might have to wait a long time before the system returns the actual results. Long waiting times can be inconvenient, but don't restrict the overall functionality. Therefore, performance is the least prioritized among all design goals.

Extensibility vs. Usability It might be desirable to extend the proposed system in the future to use different plagiarism detection technologies. Extending a system, however, also increases complexity. Another plagiarism detection algorithm might offer a different set of configuration options that are tightly coupled to its implementation. Because plagiarism detection should be reliable, easy to use, and the manual inspection as easy as possible, we prioritize usability over extensibility.

Rapid development vs. Functionality There is a limited period of 4 months to develop and integrate the proposed system in Artemis. Hence we prioritize functionality. Within the scope of this thesis, we might not implement some functionality with lower priority. We elaborate more on the status of the functionality in Section 6.1.

4.3 Subsystem Decomposition

In this section, we identify the subsystems, services and their relationship to each other. The goal in this section is to design an architecture that reduces the complexity of the system while allowing change. This can be achieved when the subsystems are loosely coupled. In Figure 4.1 the system is divided into two main subsystems: the client and the server subsystem, which are communicating through the *Submission Contents* and *Plagiarism Detection* interface.

The **Artemis server** consists of three layers, as shown in Figure 4.1. The *Web Layer* provides server resources through a REST API that is used by the client. The *Application Layer* bundles the actual application logic and contains important services, and the *Data Layer* provides connectors to the MySQL database used by Artemis. Plagiarism detection functionality is available via the *Exercise Resource*. It uses the *Plagiarism Detection Service* to start a new plagiarism detection run for a given exercise or fetch the latest results. For the latter, the *Plagiarism Detection Service* communicates with the *Plagiarism Persistence Service*, which is responsible for saving and updating plagiarism results in the database. In a new plagiarism detection run, the *Plagiarism Detection Service* uses the *Submission Service* to retrieve the students' submissions to the exercise. After comparison, the *Plagiarism Persistence Service* stores the results in the database before the *Exercise Resource* returns them via the *Plagiarism Detection* interface. The *Submission Resource* also uses the *Submission Service* to fetch the content of specific submissions directly.

The **Artemis client** consists of two layers: the *User Interface Layer* displays the components of a page and is responsible for handling user interaction. Most UI components depend on functionality provided by the *Service Layer*, which contains the client logic and communicates with the server through the presented interfaces. The *Plagiarism Detection Component* displays all elements required to start and configure plagiarism detection for a given exercise. It uses the *Plagiarism Service* to fetch existing plagiarism results or trigger a new plagiarism detection run with the configured options via the *Plagiarism Detection* interface. The *Split View Component* visualizes the results by displaying two similar submissions side by side and highlighting similarities. It requires the *Submission Service* to fetch each suspected submission's actual content via the *Submission Contents* interface and utilizes the *Plagiarism Service* to update a plagiarism incident's status.

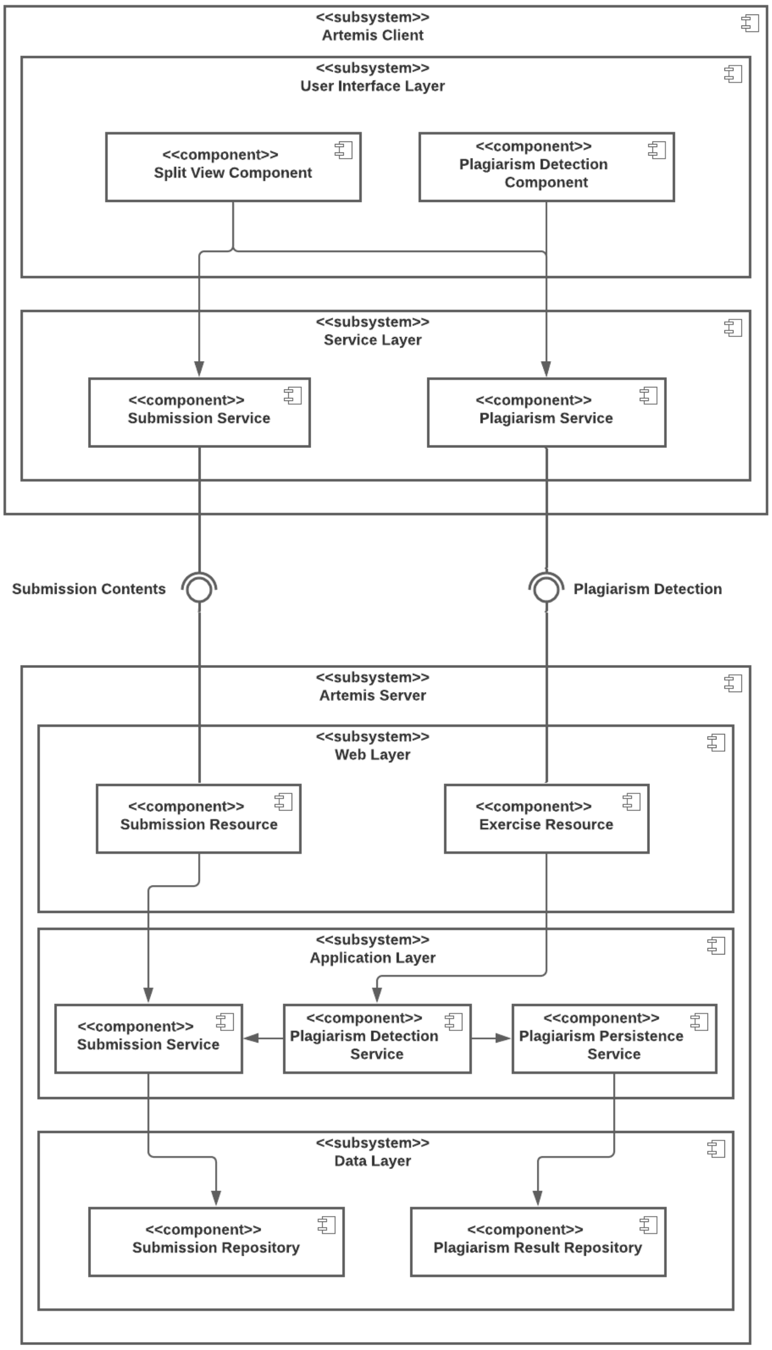


Figure 4.1: Component diagram of the plagiarism detection system following the client-server architecture of Artemis.

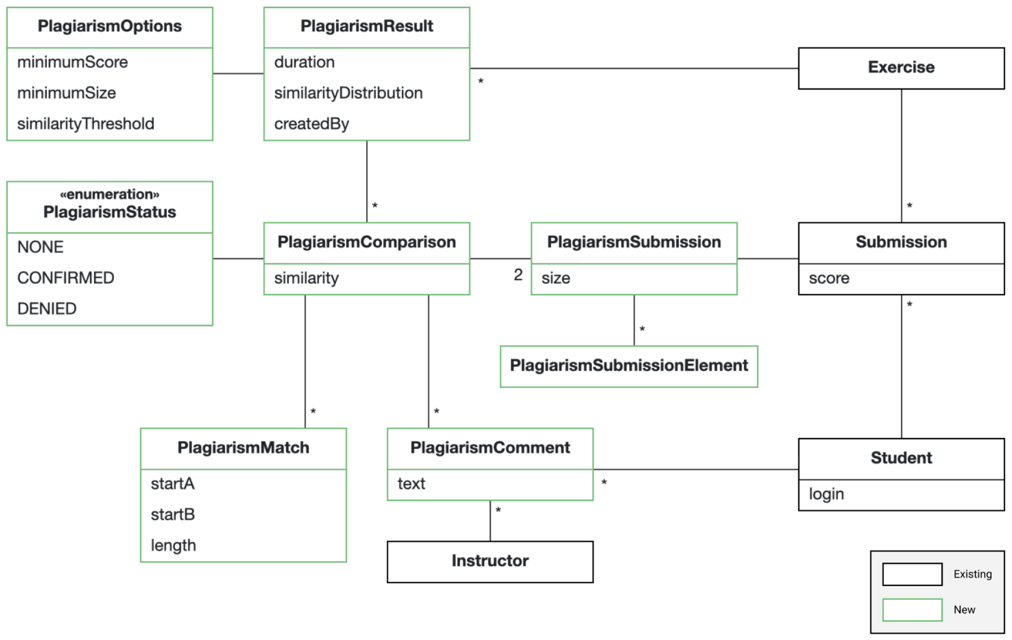


Figure 4.2: Entity diagram of the data model for plagiarism detection in Artemis

4.4 Persistent Data Management

This section explains the changes to the relational database model of Artemis required by the proposed system. We aim to relate to existing data models as often as possible to reduce the number of new tables. This design step aims to integrate new entities related to plagiarism detection without increasing the complexity of existing models and while keeping the database model extensible for future development. Figure 4.2 depicts the updated database model with relations between the new plagiarism detection entities and existing entities.

Since instructors can start automatic plagiarism detection for an exercise multiple times, each *Exercise* is in a one-to-many relationship with *PlagiarismResults*. The *PlagiarismResult* entity contains all relevant information about a specific plagiarism detection run and stores its configured options in the related *PlagiarismOptions* entity. Hence, we don't have to change the existing *Exercise* entity at all. By keeping track of who created a *PlagiarismResult* object in the *createdBy* field, other users can later consult the instructor who started plagiarism detection.

For each *PlagiarismResult*, we store all *PlagiarismComparison* objects whose similarity is above the specified similarity threshold. The *PlagiarismStatus* enumeration represents a *PlagiarismComparison's* status, which instructors can update during the manual inspection process. Each *PlagiarismComparison* relates to a pair of *PlagiarismSubmission* objects. We can extend this relationship in the future to include more than two submissions and allow the detection of group plagiarism. *PlagiarismSubmission* entities store a representation of the original *Submission* in multiple *PlagiarismSubmissionElements*, which are more efficient to compare. Again, this design allows us to keep the existing *Submission* entity unchanged.

A *PlagiarismComparison* also stores information about similarities between two *PlagiarismSubmissions* inside *PlagiarismMatch* entities. They contain the indices of similar *PlagiarismSubmissionElements* and the length of matches. *Students* and *Instructors* can leave comments on a *PlagiarismComparison*, which stores *PlagiarismComments* in a one-to-many relationship.

4.5 Access Control

This section explains the rationale behind the plagiarism detection and feedback system's access control and permission concept. Since the plagiarism system provides insight into many student submissions and the information about a plagiarism offense is highly confidential, we must strictly regulate its access. The following permission concept adheres to the *Principle of Least Privilege*, which states that every user of the system should operate using the least set of privileges necessary to complete the job [SS75].

- Overall, only **instructors** have the right to start a plagiarism detection and view all results. In addition, they can confirm or reject any detected plagiarism and optionally leave feedback, as depicted in Table 4.1.
- **Students** can only view their own incident, if any. Thus, they can see both their own and the other involved submission. In addition, the student can read feedback on their own incident and respond to it.
- **Tutors** do not have access to the plagiarism system at all in order to reduce the number of authorized persons to a minimum. This decision is driven by the Principle of Least Privilege mentioned above.

	Plagiarism Results		Plagiarism Status		Plagiarism Comments	
	READ	WRITE	READ	WRITE	READ	WRITE
Instructor	✓	✓	✓	✓	✓	✓
Tutor						
Student	✓		✓		✓	✓

Table 4.1: Access control matrix of the plagiarism detection and feedback system

Chapter 5

Object Design

This chapter describes the new JPlag API and its integration into the Artemis system in more detail. As explained in Chapter 2, before this thesis, JPlag was only available as a CLI tool and difficult to use in external software. JPlag also automatically generated web pages of the results. As this was the only form of output, post-processing the results with software would bring a significant overhead by parsing the web pages to access relevant information. Section 5.1 presents the results of our refactoring and the development of the new JPlag API. Section 5.2 discusses how we use the new API to integrate JPlag into the Artemis server.

5.1 JPlag API

Our main goal was to extend JPlag by new classes that allow other tools like Artemis to invoke JPlag in a more object-oriented way. We did not change the algorithm JPlag uses to compare submissions, and instructors can still use it from the command line. In our refactoring, adding new classes served the sole purpose of making previously unavailable functionality accessible to external programs.

As depicted in Figure 5.1, we designed the new JPlag API around three fundamental classes:

- **JPlag:** This is the main class responsible for preprocessing the submissions and performing plagiarism detection. The *JPlag* class bundles the entire business logic and exposes a simple *run* method to start plagiarism detection.

- **JPlagOptions:** Previously, users could configure JPlag via command-line options. The *JPlagOptions* class is an object-oriented representa-

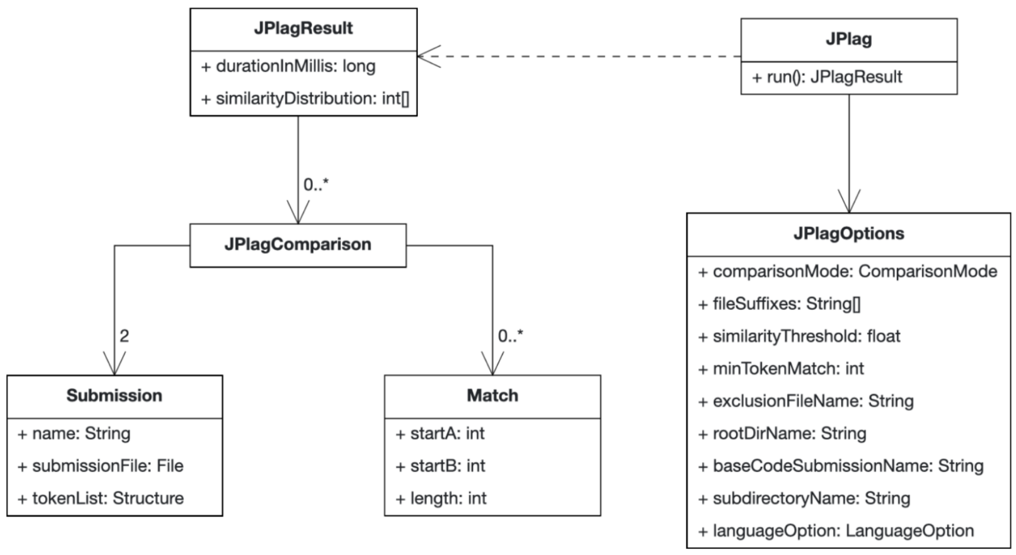
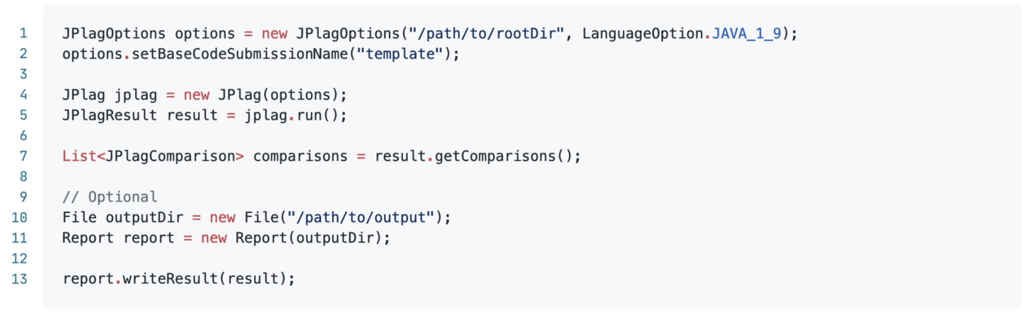


Figure 5.1: Diagram of relevant JPlag classes and their relationship after the refactoring

tion of these options. Users can create an instance of *JPlagOptions*, change its attributes according to their preferences, and pass it to a *JPlag* instance before starting the plagiarism detection.

- **JPlagResult:** To make the plagiarism detection results easier to interpret by other programs, we created the *JPlagResult* class containing all relevant information about a plagiarism detection run. This class allows other programs to post-process the results, store them in a database, or export them in a custom format. Generating a report from a *JPlagResult* is optional.

Figure 5.2 shows an example usage of the new JPlag API. In lines 1-2, we create a new *JPlagOptions* instance with the path to the directory containing all student submissions and the programming language of the source code files. We also specify the name of the directory with the template code that all submissions are based on. In line 4, we instantiate a *JPlag* object and pass the *JPlagOptions* instance to configure the program. Then, we call the *run* method to parse the submissions and perform the actual plagiarism detection. The *run* method returns a *JPlagResult* instance that we use in line 7 to get the list of all comparisons. We could access additional information about the plagiarism detection run or post-process the *JPlagComparion* objects at this



```
1   JPlagOptions options = new JPlagOptions("/path/to/rootDir", LanguageOption.JAVA_1_9);
2   options.setBaseCodeSubmissionName("template");
3
4   JPlag jplag = new JPlag(options);
5   JPlagResult result = jplag.run();
6
7   List<JPlagComparison> comparisons = result.getComparisons();
8
9   // Optional
10  File outputDir = new File("/path/to/output");
11  Report report = new Report(outputDir);
12
13  report.writeResult(result);
```

Figure 5.2: Example code snippet of the new JPlag API

point. In lines 10-13, we optionally create a new *Report* of the *JPlagResult*, which we can export to generate web pages that visualize the results.

5.2 Integration of JPlag in Artemis

In Artemis, we use JPlag to detect plagiarism for programming and text exercises. On the server, the *Plagiarism Detection Service* interfaces with the new JPlag API. Figure 5.3 shows a more detailed component diagram of the Artemis server's application layer, where we can locate the JPlag integration.

When the client sends a request to the server to detect plagiarism for a given exercise, the *Plagiarism Detection Service* is responsible for performing or delegating the actual comparison of submissions and returning the results. In the case of programming and text exercises, the *Plagiarism Detection Service* will delegate plagiarism detection to JPlag. Before, it prepares the submission to be compared by JPlag. Therefore, the service downloads the submissions and writes them to the local file system on the server. All submissions must be in the same folder, referred to as the root directory.

We can then invoke JPlag similar to how Figure 5.2 does. JPlag requires the path to the root directory and the language of the submissions. Parsing the submissions and performing plagiarism detection is taken care of by the internal JPlag classes, which return the plagiarism detection results to the *Plagiarism Detection Service*. We map the results to a more generic representation of plagiarism results compatible with results for modeling exercise. As seen in Figure 4.2 from the previous chapter, the database scheme is very similar to JPlag's class hierarchy presented in Figure 5.1. This similarity exists by design and makes it easy to persist the results.

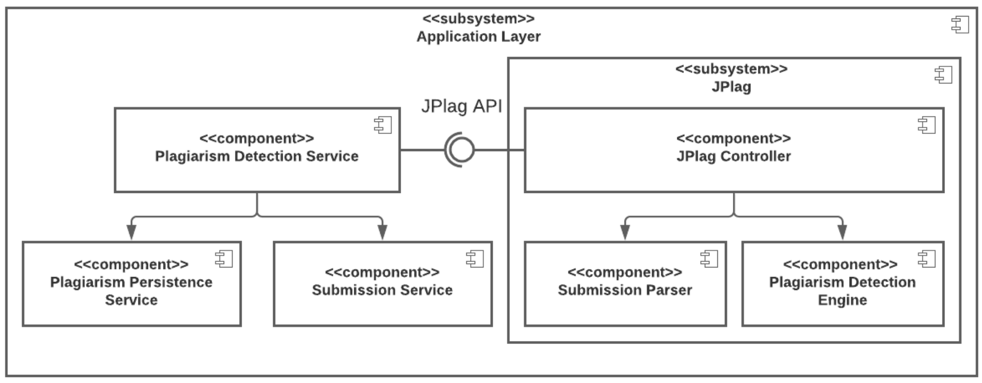


Figure 5.3: Detailed component diagram of the Artemis Server's Application Layer

Chapter 6

Summary

This chapter summarizes the results of this thesis by discussing the realized and open goals in Section 6.1. Following this we conclude the results of this thesis in Section 6.2 and formulate an outlook on future work that could proceed this thesis in Section 6.3.

6.1 Status

In this section we compare the status of the system with the requirements we have elicited in Chapter 3. We divide the requirements into the following three categories:

- ● **Implemented Requirements:** The requirement is realized and no further changes are required.
- ◐ **Partially Implemented Requirements:** The requirement is only partially implemented and further changes are required to have the requirement in a state that will be accepted by the client.
- ○ **Not implemented Requirement:** The requirement is not fulfilled as it is not implemented yet.

The status of the functional requirements is summarized in Table 5.1 and the status of the nonfunctional requirements can be found in Table 5.2.

6.1.1 Realized Goals

In total, we managed to implement 10 of the 18 functional and nonfunctional requirements elicited in Section 3.3. Specifically, we introduced a new page

Functional Requirements		Status
FR1	Compare submissions side by side	●
FR2	Manage detected plagiarism matches	●
FR3	View plagiarism statistics	●
FR4	Highlight similarities	●
FR5	Configure automatic plagiarism detection	●
FR6	Add detailed feedback	○
FR7	Respond to feedback	○
FR8	See plagiarism detection progress	○
FR9	Learn from manual inspection	○
FR10	Detect group plagiarism	○

Table 6.1: Implementation status of the functional requirements (● completed - ◐ partially completed - ○ incomplete).

for plagiarism detection to the Artemis client, which instructors can use to start and configure a new plagiarism detection run (FR5). After comparison, it displays suspected submissions side by side (FR1) and highlights similarities between them by color, making results more efficient to compare (FR4). Instructors can also manage a plagiarism incident's status (FR2) by confirming or denying it and view details about plagiarism detection runs like their duration or the similarity distribution of the results (FR3).

As depicted in Table 6.2, the developed system (partially) fulfills 7 of the eight nonfunctional requirements. The new plagiarism detection page allows instructors to start and configure automated plagiarism detection both from one site (NFR1). Because the split view panes are resizable and instructors can collapse the configuration panel to maximize the split view, manual inspection of the results works on tablet and desktop devices (NFR2). The sidebar gives instructors easy access to single plagiarism incidents. Because the list of plagiarism incidents is paginated and only displays a maximum of 100 items at once, the client application stays responsive even for large results (NFR7). Clicking on a sidebar item loads the suspected submissions' contents, and instructors can update the incident's status with one click on the corresponding buttons in the split view header (NFR3). Plagiarism results don't change if submissions and configuration stay the same, but we

Nonfunctional Requirements		Status
NFR1	Ease of Use	●
NFR2	Responsiveness	●
NFR3	Efficiency	●
NFR4	Consistency	◐
NFR5	Robustness	○
NFR6	Response Time	◐
NFR7	Load Time	●
NFR8	Extensibility	●

Table 6.2: Implementation status of the nonfunctional requirements (● completed - ◐ partially completed - ○ incomplete).

don't have data of the number of student responses on plagiarism accusations yet, leaving *Consistency* (NFR4) partially fulfilled. NFR6 is also only partially fulfilled because plagiarism detection achieves the desired response time only for modeling and text exercises. Last, the developed system is extensible and can support new plagiarism detection algorithms or configuration options (NFR8).

6.1.2 Open Goals

Due to the limited time available to develop the proposed system, we didn't fully implement 5 of the functional and 3 of the nonfunctional requirements. Although instructors can update the status of a plagiarism incident, it is not possible yet for instructors to leave feedback on that incident for the involved students (FR6), as shown in Table 6.1. As a consequence, students also cannot reply to any feedback directly on Artemis (FR7). Hence, instructors must still use traditional means of communication like e-mail to contact the students. We also didn't manage to implement a progress bar for plagiarism detection with information about the pending comparisons (FR8). Only a loading spinner is showing up while plagiarism detection is in progress. The time restrictions also didn't allow us to fulfill the more visionary requirements of learning from manual inspection results to improve automated plagiarism detection (FR9) and detecting group plagiarism (FR10).

In terms of nonfunctional requirements, we didn't achieve *Robustness*

(NFR5) yet, which is a direct consequence of the missing feedback system for plagiarism incidents. *Consistency* (NFR4) and *Response Time* (NFR6) are partially fulfilled, as stated in the previous section.

6.2 Conclusion

In this thesis, we successfully improved the integration of plagiarism detection in Artemis. Before, the integration was very limited and required instructors to perform numerous manual steps to process the results. We introduced a new plagiarism detection page to the Artemis client, allowing instructors to start and configure automated plagiarism detection for a given exercise.

Previously, instructors had to download the results and use external software to inspect the incidents. Now we display all detected plagiarism incidents on the plagiarism detection page. A side-by-side comparison of suspected submissions and the highlighting of similarities between them facilitates manual inspection. Although leaving feedback on confirmed plagiarism incidents is not possible yet, instructors can update a plagiarism incident's status and download the processed results to contact only students whose submissions were confirmed as plagiarized before.

Leaving communication with students the only feature not fully implemented, we managed to integrate most of the previously time-consuming tasks of plagiarism detection into Artemis and offer a more efficient workflow for instructors to inspect and manage plagiarism incidents.

6.3 Future Work

This chapter presents some ideas to further improve the plagiarism detection and feedback system in Artemis. Besides implementing the unfulfilled requirements elicited in Section 3.3, we could also develop more visionary features in the future that improve the user experience for instructors and students.

Automatically start plagiarism detection. After an exercise's due date passed, the system could automatically start plagiarism detection for the given exercise. For this, the system could choose configuration values that yield the best results on its own or use options instructors specified beforehand. Making the automated plagiarism detection process asynchronous also has the advantage that instructors can still use Artemis for other tasks while plagiarism detection is in progress.

Improve plagiarism notifications. Especially if plagiarism detection becomes an asynchronous process, it's important to notify instructors when new results are available so that they don't have to check for updates themselves. We could communicate plagiarism detection updates via Artemis' notification system. Additionally, students could receive a notification if one of their submissions was accused of plagiarism.

Extend plagiarism context. In the future, the system could detect plagiarism not only among all current submissions to an exercise but consider previous years' submissions as well. Furthermore, plagiarism detection could also compare submissions with external sources like internet articles, forum posts, or literature for similar code snippets or text fragments.

List of Figures

List of Tables

Bibliography

[Aik20] Alex Aiken. Moss, a system for detecting software similarity. `http://theory.stanford.edu/~aiken/moss/`, 2020. Retrieved: 2021-02-05.

[Bak93] Brenda S. Baker. On finding duplication in strings and software. *Journal of Algorithms*, 1993.

[BCJ+09] Amitav Banerjee, U. B. Chitnis, S. L. Jadhav, J. S. Bhawalkar, and S. Chaudhury. Hypothesis testing, type i and type ii errors. *Industrial Psychiatry Journal*, 18(2):127–131, 2009.

[BD09] Bernd Bruegge and Allen H Dutoit. *Object Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall, 2009.

[BM09] Tracey Bretag and Saadia Mahmud. A model for determining student plagiarism: Electronic detection and academic judgement. *Journal of University Teaching & Learning Practice*, 6(1), 2009.

[BM21] Eren Bilen and Alexander Matros. Online cheating amid covid-19. *Journal of Economic Behavior & Organization*, 182, 2021.

[BRC19] Imene Bensalem, Paolo Rosso, and Salim Chikhi. On the use of character n-grams as the only intrinsic evidence of plagiarism. *Language Resources and Evaluation*, 53:363–396, 2019.

[CL01] Fintan Culwin and Thomas Lancaster. Plagiarism, prevention, deterrence and detection. *Higher Education Academy*, 2001.

[Dic89] Oxford English Dictionary. *Plagiarism*. Oxford University Press, 1989.

[HZ03] Tim Hoad and Justin Zobel. Methods for identifying versioned and plagiarized documents. *Journal of the American Society for Information Science and Technology*, 54(3), 2003.

[KS07] Dominic Keuskamp and Regina Sliuzas. Plagiarism prevention or detection? the contribution of text-matching software to education about academic integrity. *Journal of Academic Language & Learning*, 1(1):91–99, 2007.

[KS18] Stephan Krusche and Andreas Seitz. Artemis: An automatic assessment management system for interactive learning. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 284–289, 2018.

[KvFA17] Stephan Krusche, Nadine von Frankenberg, and Sami Afifi. Experiences of a software engineering course based on interactive learning. *SEUH*, 2017.

[MST95] Donald Michie, David J. Spiegelhalter, and Charles C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1995.

[PMP00] Lutz Prechelt, Guido Mahlpohl, and Michael Philippsen. Finding plagiarisms among a set of programs with jplag. *Journal of Universal Computer Science*, 8(11), 2000.

[RC07] Chanchal Kumar Roy and James R. Cordy. A survey on software clone detection research. *Technical Report*, (2007-541), 2007.

[SB75] R.J. Smith and R.G. Bryant. Metal substitutions in carbonic anhydrase: A halide ion probe study. *Biochemical and Biophysical Research Communications*, 64(4), 1975.

[Smi18] Katherine Smith. University bosses call for ban on essay-writing companies. `https://www.bbc.com/news/education-45640236`, 2018. Retrieved: 2021-02-01.

[SS75] J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[SWA03] Saul David Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: Local algorithms for document fingerprinting. 2003.

[Tur] Turnitin. The plagiarism spectrum. `http://go.turnitin.com/paper/plagiarism-spectrum`. Retrieved: 2021-02-01.

[V.S97] Stephen V.Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*, 62(1):77–89, 1997.

[Wis93] Michael J. Wise. String similarity via greedy string tiling and running karp-rabin matching. 1993.