The Grader: A Grading Assistant for Lab Tests and a Teaching Tool

Anonymous Authors
Anonymous Organization

***Abstract*—This article presents the design and implementation of the Grader, a grading assistant application deployed for a Web Application Development course at our school. The Grader is equipped to handle various logistical aspects of lab tests, including file management, consistent application of rubrics, and auto-grading of questions with test cases. Additionally, it incorporates heuristic rules to detect cheating attempts. We anticipate that the Grader will find widespread utility in programming courses where lab tests serve as summative assessments. Developed within the same programming environment taught in the class, the Grader also serves as a pedagogical tool, demonstrating to students a substantial project that utilizes the techniques learned in the classroom. This article describes the features of the Grader, the context of the course where it is deployed, and our experience and insights in using it. In particular, we will present the comments from the students gathered from their qualitative feedback on the course and the instructor as the utility of the Grader in enhancing their learning experience. We will also present the estimates of the time saved by the instructors in evaluating lab tests. From the data collected, we conclude that the Grader can serve as a pedagogical too to motivate our students. Further, the Grader can significantly reduce the demand on the instructors time in grading lab tests. In terms of detecting plagiarism, its performance is not fully established.**

***Index Terms*—grading assistant, programming courses, lab tests, web application development, automatic grading**

I. Introduction

Lab tests are a commonly used assessment technique for programming courses [1], [2]. They focus on evaluating a student's programming skills and understanding of the course material through hands-on exercises. One programming course that has become an integral part of the undergraduate curriculum of information systems is Web Application Development (WAD) [3]–[5]. In our school, we offer the first of two courses in WAD to our freshmen-year, second-term students. The course also introduces them to object-oriented methodologies. As is common with courses dealing with programming languages, we have lab tests as part of its summative assessment.

When compared to other modes of testing (such as online, e-learning mode, or the traditional hard-copy mode), lab tests present their own challenges when it comes to grading, related to file management and grading-data handling. In order to tackle such challenges, we developed a tool, which is presented in this paper. We will refer to the grading tool as the "Grader."

In addition to helping the instructors with the logistics of grading, the development of the Grader is also an opportunity to introduce our students to a real-world application using the technologies they are taught in class. It has the added attraction of dealing with an issue close to their hearts, namely their grades, and exposes them to some of the more advanced ideas in object-oriented approach.

Lastly, since the Grader stores all the student submissions in a database, we can perform deep analysis on their answers to detect cheating attempts. This is especially important in lab tests because students have access to their computers during the tests. Although they are warned not to connect to the network, nor attempt to use any messaging applications, we can never be certain about their academic integrity despite our stringent invigilation. Manual detection, therefore, is our last line of defense against cheating attempts. The Grader can make this process by automating several of the pairwise comparisons of student answer books. Currently implemented using a heuristic algorithm, the cheating detection can easily be extended using text analytics and techniques specifically tailored for source code plagiarism [6] such as JPlag, MOSS etc.

In this paper, we seek to answer the following questions using primarily qualitative data such as student comments and instructor feedback:

- **RQ1**: Can we reduce the time demand on instructors in grading lab tests using automation?
- **RQ2**: Can we motivate students of programming courses by presenting a real-world application?
- **RQ3**: How well can heuristic comparisons discourage and detect plagiarism in lab tests?

This article is organized as follows: We begin by examining the related work in the areas of web application development courses, lab exams in programming courses, and grading assistants. Additionally, we explore the current state of heuristic plagiarism detection and automated grading techniques. Next, we present contextual information about our specific course, including details about the lab tests conducted and the demographics of the students involved. Subsequently, we delve into the design goals of the Grader and the features implemented. Furthermore, we share our experience with the Grader, discussing its effectiveness and usability. We also outline our future plans and potential enhancements for the application. Finally, we conclude the article, summarizing the key findings and insights gained from our use of the Grader in the assessment of lab exams for web application development courses.

II. Related work

Over the past few decades, Web Application Development (WAD) has emerged as a fundamental component of undergraduate information systems and computer science education. Extensive research has been conducted to emphasize the importance, relevance, and challenges associated with WAD [7], [8]. Since WAD courses, particularly within information systems departments, primarily focus on programming [9], they share many pedagogical and assessment techniques with programming language courses [10]. One prominent technique used for summative assessment in programming courses is the lab test [1].

As enrollment in information systems and computer science programs continues to rise, the demand for testing, assessing, and grading students also increases. Automatic grading for programming courses has garnered significant attention in the literature. A recent survey by Paiva et al. [11] explores the current state of the art, highlighting the challenges and opportunities in this area. Various articles propose automated and language-agnostic testing frameworks for formative assessments [12], [13]. In a different study, Cohenour et al. [14] employed MATLAB as a grading tool, utilizing ActiveX Com controls to provide feedback messages via email, making it suitable for formative assessment, albeit limited to the Windows operating system.

Moving beyond traditional programming courses, there have been endeavors to automate grading assignments and tests in spreadsheet-based courses [15]. These initiatives primarily focus on formative assessments and describe frameworks that enable instructors proficient in Excel and VBA to develop automatic graders. Thulasidas [16] discusses an iterative approach for building and applying grading rubrics in summative assessments. In their comprehensive study, Weegar et al. [17] explore techniques for automating the grading of short-answer questions and argue that a fully automated grading solution remains elusive. While the Grader application presented in this article incorporates auto-grading functionality, its primary objective is to assist instructors in consistently applying grading rubrics and managing grading logistics, including file management and score tracking.

Ensuring academic integrity in tests and assessments is always a concern, and traditional methods often rely on strict invigilation. However, this approach becomes unreliable in the case of lab tests as students have access to their laptops and network connectivity for at least part of the exam duration. While some studies [18] suggest that instances of academic dishonesty are not more prevalent online than in traditional settings, it remains the responsibility of instructors to ensure integrity and fairness. Cluskey et al. [19] proposed eight control procedures for online examinations, which were not directly applicable to our summative assessments based on lab tests because they were primarily aimed at minimizing proctoring costs. Another approach, suggested by McHaney [20], is to foster an awareness and culture of academic integrity. Additionally, research has explored video monitoring as a means of migrating the traditional monitoring strategy to the online world [21].

Another strategy for detecting cheating attempts is to focus on student submissions and compare them using either machine learning techniques [22] or heuristic rules, which is the approach employed in the Grader application. Alternative approaches include personalizing questions so that each student receives a unique version [23], as well as integrated systems [13], [24] that handle all aspects of assignments and tests, from conception to administration and grading. The approach adapted in the Grader is different from what is seen in the literature. It limits its scope to helping instructors the grading chores, and flagging potential cheating attempts.

III. Course and Context

A. Web Application Development

Our courses on Web Application Development (WAD) are designed to teach students the necessary skills and knowledge to develop dynamic and interactive websites and web-based applications. Offered over two successive terms, they focus on the principles, tools, and technologies used in building web applications and provides hands-on experience in creating functional and user-friendly web-based systems.

The grading tool described in this article is developed for the first of the two WAD courses, which we will refer to as WAD1. In this course, students learn the following:

1) **Web Technologies:** Students are introduced to HTML (Hypertext Markup Language), including tables and forms and their basic styling.
2) **Backend Development:** Students learn PHP as a server-side programming language, enabling the development of the server-side logic that handles form and data processing, and business logic for web applications.
3) **Database Integration:** Our students concurrently learn SQL (Structured Query Language) in a different course. In WAD, they learn how to integrate databases into web applications.
4) **Object-Oriented Programming:** In addition to WAD-specific topics, our students also get their first exposure to object-oriented programming in WAD1.

Throughout the course, our students solve in-class exercises (ICE) to apply the concepts and skills they have learned. We use WampServer on Windows and MAMP on MacOS for them to work on ICE. They install these packages on their laptop and launch the Web and MySQL servers on `localhost`. They develop their code using the popular Visual Studio Code editor and use their personal laptops to run them. By the end of the course, students have a solid understanding of how the web works, and are familiar with web-development principles. They also possess the basic skills needed to continue learning and adapting to new web technologies as they evolve.

More advanced topics such as frontend development (Javascript frameworks, CSS styling), web application architecture (MVC or RESTful APIs), project management, testing,

TABLE I: Contextual Information of the Cohort

Cohort Year	First Year
Number of Students	176
Number of Sections	4
Students per Section	44, 43, 44, 45
Male:Female Ratio	125:51
Delivery Mode	Face-to-face

deployment etc. are taught in the second course in WAD as well as in other courses offered by our school.

B. Contextual Information

The first part of our two WAD courses is offered to the first-year, second-term undergraduate students in information systems, with the second part offered in the following term. Our students are already exposed to basic programming in Python before taking WAD1. In addition, they take a course on database management concurrently with WAD1. As is common in STEM programs, our cohort is predominantly male. The details of the cohort (where the Grader was deployed) are tabulated in Table I. Our e-learning platform is based on Brightspace [25], and we will refer to it as eLearn in this article.

C. Lab Tests

For our WAD1 course, we have two lab tests, where students are provided with specific programming tasks and problems to solve within a given time frame. These tasks are designed to assess the students' ability to apply the programming concepts, algorithms, data structures, and problem-solving techniques they have learned during the course.

The scores obtained in lab tests are used to assess a student's performance and understanding of the programming concepts taught in the course. They contribute to the overall evaluation of a student's progress and are combined with other assessment components, such as quizzes and exams, to determine the final course grade. The lab test components carry 50% weightage in the overall assessment of this WAD1 course.

In order to get them started with the lab tests, we provide the students with the templates or skeleton solutions that we call the "Resources." The students are expected to complete the resource files and upload them to our e-learning platform. Each lab test may have 10 to 20 resource files. The provision of the Resources helps standardize the files submitted by the students as well.

For grading, the instructors have the completed versions of Resources, which we will call the "Solutions." Each student will submit their own updated versions of the Resources, which we will refer to as "Answers" in this article. The process of grading essentially is in comparing the Answers with the Solutions, and assigning scores with the use of rubrics.

D. Rubrics and Automated Grading

In our lab tests for WAD1, we have questions of different degrees of difficulties. While most of the questions are graded manually by the instructors, about 30% of the lab test is graded automatically using automated test cases and scripts. They work by running the student's code against a set of predefined test cases and evaluating the correctness and functionality of students' code based on whether it produces the expected outputs or exhibits the desired behavior.

As is common among instructors, we also have rubrics and scoring guides that define evaluation criteria and provide clear guidelines for grading our lab tests. They establish a framework that outlines the expectations and standards for students' work, making the grading process as consistent and transparent as possible.

TABLE II: Workflow of the Lab Tests for our WAD courses

No.	*Step*
1	Design the questions and sub-questions for the test. Create the Resources (the skeleton solutions that the students will modify) and the Solutions (the reference solutions the instructors will use for grading).
2	Prepare grade book on the e-learning platform (which we will refer to as eLearn) by creating the grade items per sub-questions and questions in the lab test.
3	Download the empty grade books as CSV templates for future upload back to eLearn.
4	Prepare the rubrics. Typically, each sub-question will have a set of rubrics as scoring guides.
5	Prepare the test cases for questions that are to be auto-graded.
6	Administer the lab tests. • The students download the Resources. • They then turn off their WiFi connection. • They complete the files as per the instructions and requirements of the questions. • After the test, they create a Zip archive of their edited versions of the Resources. • They turn on their WiFi and upload the Answers (as one Zip file) to eLearn.
7	Download the student Answers as Zip archives.
8	Unpack and validate the Answers. If some of the archives are invalid, it needs to be addressed quickly so that the students can rectify the issue.
9	Grade the Answers by comparing each file to the corresponding Solution and applying the rubrics.
10	Generate and examine grading statistics. If the score distributions are not as expected, regrading may be necessary.
11	Detect cheating attempts, if possible, by comparing student Answers. In practice, this step becomes too onerous to implement.
12	Prepare CSV files and upload scores to eLearn.

E. Grading Challenges

Although consistent use of rubrics can assist the assessment, lab tests typically come with set of challenges from the perspective of the instructors grading them.

1) One of the challenges of grading a lab test is the necessity to keep track of multiple files per student. While technically not complicated, in practice, the existence of a large number of files with identical names becomes a logistical nightmare.

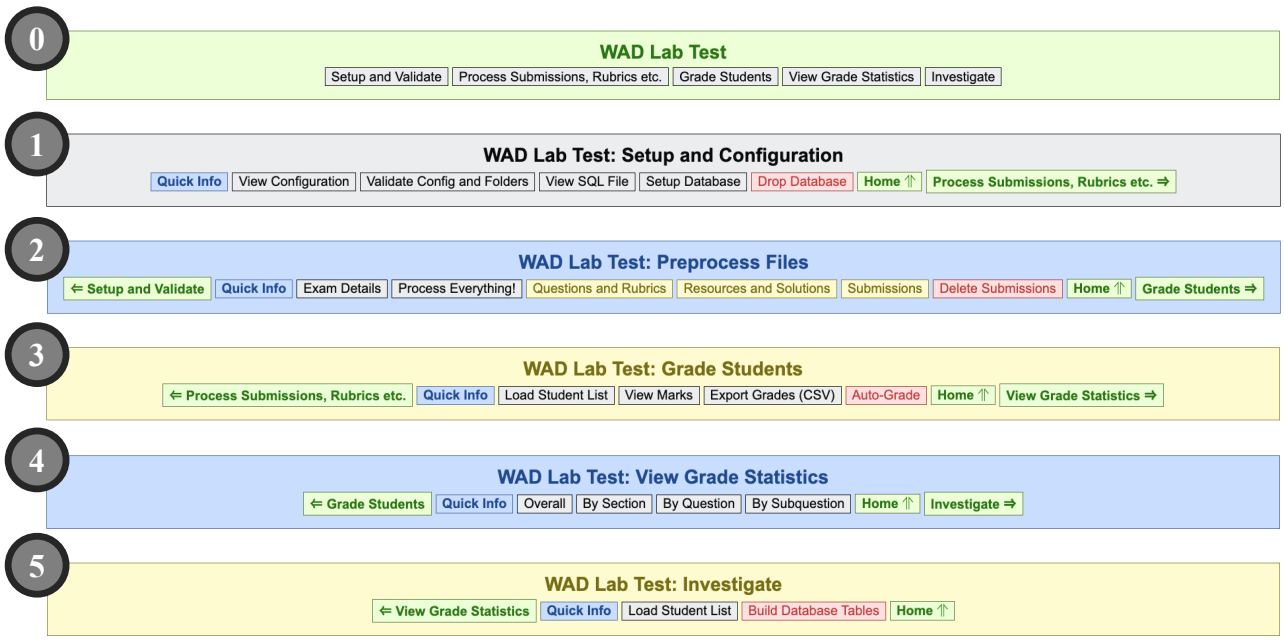

Fig. 1: The menu structure of the Grader application. (0) is the welcome page, showing the buttons arranged in the logical sequence of a typical grading workflow. Clicking on each button brings up another page with its own menu to complete the corresponding task, each of which has help (**Quick Info**) button and navigation buttons to next/previous task.

2) Often, the instructors may have to debug the submitted files so that they can run and produce some output without any fatal errors. The need to keep track of the debugged versions of the file exacerbates the already difficult logistical challenge.
3) The data entry requirement of the scores also is taxing. For each question, and on a per-student basis (often on a per-file basis), instructors have to enter scores after aggregating over the rubrics of the question. Although we try to minimize the effort needed by designing clever spreadsheets, we still waste significant time and effort on this aspect of grading.
4) Another challenge is in the format and content of the uploaded files. We require the students to upload a Zip archive of their edited versions of the Resource files. We expect to see a `resources` folder as the root of the archive. But often, students creates archives out of the parent folder (or parent's parent, or even several levels above), requiring us to standardize their submissions.
5) We also have students submitting empty archives of the shortcut or alias of the Resource files, which may make it necessary to accept their emailed files well after the official deadline for the submission.
6) Since the files are soft copies that can be shared despite our careful invigilation, the detection of cheating attempts also becomes a daunting challenge while grading.

Considering each of these challenges as an opportunity to innovate, we designed the Grader explicitly to address them, as we will describe in the following sections.

F. The Lab Test Workflow

Since the Grader application is designed to assist the grading steps of our lab tests, it is necessary describe the workflow in administering and grading our lab tests for WAD. For each of the two lab tests we have for WAD1, we have the sequence of steps as shown in Table II. The Grader helps with the items after the administration of the tests, namely steps 7 through 12. The items in the workflow are likely to be similar in the lab tests of other programming courses, and the Grader application may be deployed for them as well.

IV. The Grader Application

Over the course of its development, the Grader has grown into a mature web application. Although we will not describe its architecture, we will go through the features of this menu-driven application in some detail.

The Grader follows a menu system using buttons, as shown in Fig. 1. When it is launched (typically as a web application running on the `localhost`), it displays the menu labelled (0) in Fig. 1. On this page, instructors click the buttons roughly from left to right to complete their grading workflow. Each button brings up another page with a set of action buttons labelled (1) through (5) in Fig. 1. In the following description, we will go through these features and the tasks that the Grader can handle.

A. Grader Configuration

Before launching the Grader, it is necessary to configure it. The configuration is driven by a single php file (`config.php`), as is common with most large-scale web applications. In addition to editing this configuration file, we have to specify the

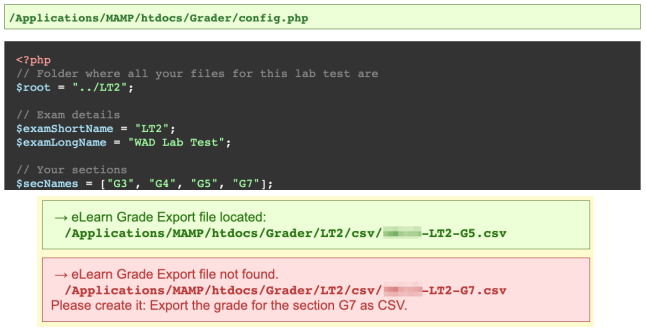

Fig. 2: Viewing the configuration file and validating it.

questions, subquestions, rubrics, the Resources and Solutions so that we can populate the respective database tables.

Based on the principle of single source of truth (whereby we enter information only once, in one system), we follow the steps below to configure the Grader:

1) Place the reference files (from Table II, steps 1) in a location accessible by the Grader and specify it in `config.php`.
2) Place the grade book files (Table II, steps 2 and 3) and in a location and specify it in `config.php`.
3) Create the rubrics file `rubrics.csv` by hand, (Table II, step 4) and specify its location in `config.php`.
4) Specify the question to be auto-graded and the test cases (Table II, step 5), also in `config.php`.
5) After the lab test, download the student Answers as Zip archives (Table II, step 7) in a location and specify it in `config.php`.

Once these configuration steps are done, we can use the Grader to handle the rest of the workflow items (Table II, steps 8 to 12). We do that by following the menu system (Fig. 1) as described below:

(1) Setup and Validate: This first step helps read and validate the configuration file. It also has actions to set up the database tables and prepare the Grader for further processing. The buttons in this menu are shown in Fig. 1, labelled (1) and perform the following tasks:

View Configuration: View (not edit) the current settings in the `config.php` file, as shown in Fig. 2.

Validate Config and Folders: Validate the configuration in `config.php` file and check for the existence an contents of the folders and files specified. (See Fig. 2.)

View SQL File: Examine the SQL file that will be run to set up the database. It should not be edited because it is generated based on `config.php` and a template.

Setup Database: Run the generated SQL file to set up the database. Clicking the button multiple times is harmless; it does not reset the database.

Drop Database: Drop all database tables and restart: This button will really reset the database. If we have graded some students, all the entered grades will disappear. The database operations are shown in Fig. 3.

(2) Process Submissions, Rubrics etc.: In the second step, we populate the database tables by importing `rubrics.csv` and the grade books downloaded from eLearn to create and populate the database tables `questions`, `subquestions` and `students` tables. The interface is shown in Fig. 1, labelled (2), with the following buttons:

Exam Details: Displays (as in Fig. 4) the lab test information, as imported from the grade books.

Process Everything! A shortcut to do what the next three buttons do.

Questions and Rubrics: Import grade books (`CSV` files downloaded from eLearn) and `rubrics.csv` to create questions, subquestions, and their rubrics in the database.

Resources and Solutions: Process the Resource and Solution files into the database.

Submissions: Import student submissions to create Answers and prepare for grading. The Grader will verify that students have submitted valid zip files, and produce color-coded status messages per student and a summary for the cohort, as shown in Fig. 5.

Delete Submissions: Delete the uncompressed answer files that the student have submitted. It does not delete the zip files downloaded from eLearn.

(3) Grade Students: The next phase is to grade the students. Clicking on this button will make the process as painless as possible by providing a page with the following action buttons (as shown in Fig. 1, labelled (3)).

Load Student List: Load the list of students to grade each one. The list is color-coded to indicate the students who

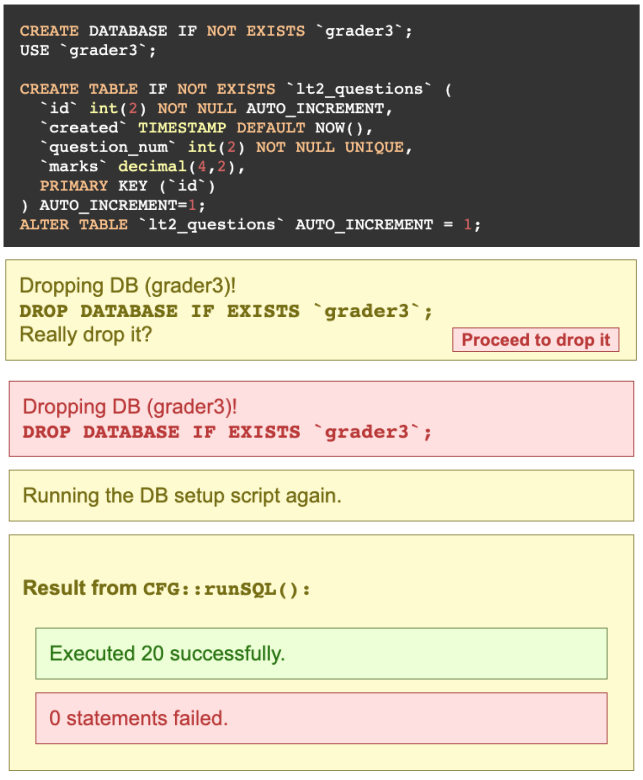

Fig. 3: Viewing the database schema, importing it and deleting the tables.

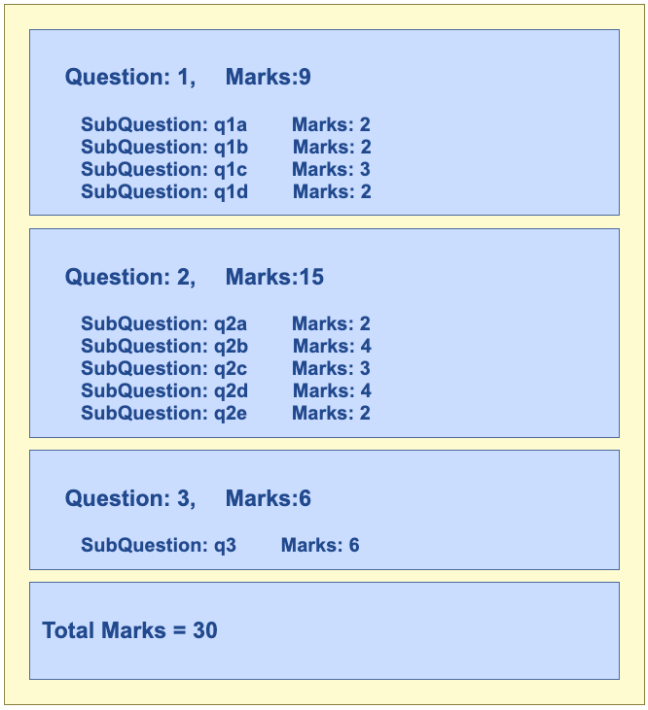

Fig. 4: Viewing exam details.

are absent for the test, being graded or fully graded. Once loaded, a student can be graded by clicking on their **Grade**, **Continue** or **Regrade** button, as shown in Fig. 6. Clicking on this button will bring up the grading window (Fig. 7) with color coding (green for full marks, yellow for partial marks and red for zero or penalties). The grading window also has navigation buttons near the top as well as a button to

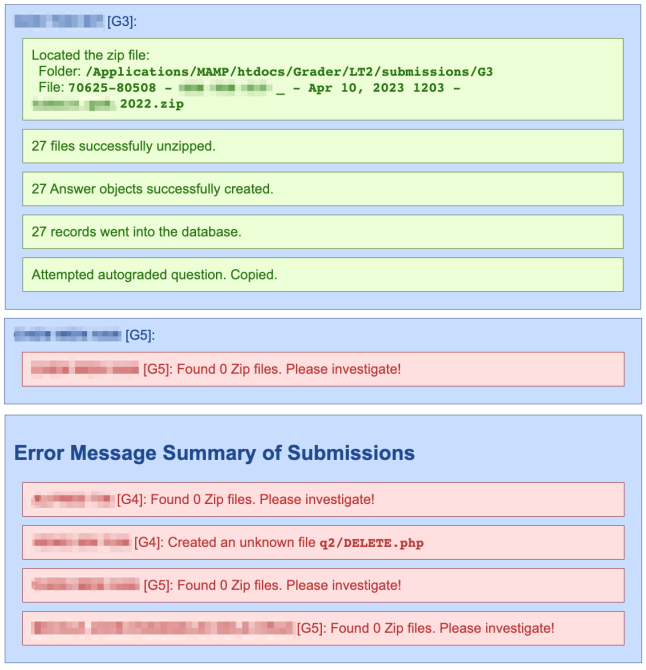

Fig. 5: Importing student submissions and validating them.

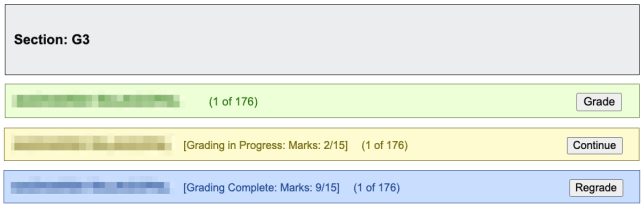

Fig. 6: List of students with the current grading status indicated and color-coded, with context-sensitive action buttons.

view the student's Answer files. Once clicked, it shows the answer file with an option to insert a grading comment, and to view its output running on the `localhost` server (Fig. 8). Note that the Answer files viewed are always the submitted ones (from the database), while the output shown is from the edited ones (from the disk) in case the instructor needs to make debugging modifications.

View Marks: This button is to display the marks in a nice table form that will be exported when the **Export CSV** button is clicked, as in Fig. 9.

Export Grades (CSV): After the grading process is completed, student scores are exported to a CSV file (which can be imported to eLearn with minimal modifications) using this button. The file names will be `Export-{gradebook}.csv` where `{gradebook}` is the name of the grade book exported from eLearn. The files will be in the same folder as the grade book exports, and can be imported back to eLearn.

Auto-Grade: If auto-gradable questions are specified in `config.php`, this button will run the test cases for all stu-

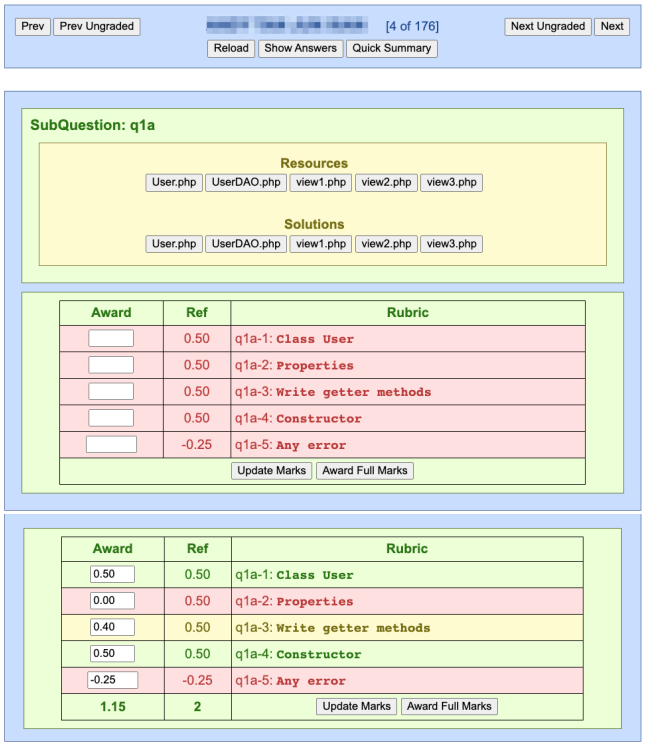

Fig. 7: The grading interface.

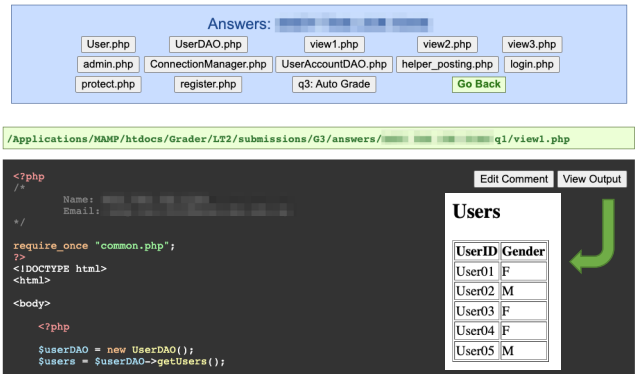

Fig. 8: Inspecting student Answer file.

Section: G3					
	q1a → 2	q1b → 2	q1c → 3	q1d → 2	q3 → 0.00
	q1a → 2	q1b → 2	q1c → 3	q1d → 2	q3 → 0.00
	q1a → 2	q1b → 2	q1c → 2.95	q1d → 2	q3 → 0.00
	q1a → 2	q1b → 2	q1c → 2.65	q1d → 2	q3 → 0.00
	q1a → 1.5	q1b → 1.5	q1c → 1.5	q1d → 1.5	q3 → 0.00
	q1a → 2	q1b → 2	q1c → 3	q1d → 2	q3 → 0.00

Fig. 9: Viewing marks.

dents. It will update the database with the marks computed. The auto-grader file should output the total marks in some recognizable form, as specified in `auto-grade` options in `config.php`. The options should also specify the subquestion name and the rubric name, so that the marks can be inserted into the database. For those students whose file crashes, the Grader will put zero in the database, but will display buttons to view and manually grade the question.

(4) View Grade Statistics: Once the grading process is complete, the next logical step is to study the statistics. The Grader can compute and display several statistical measures of the grade distributions using the fourth menu in Fig. 1. As shown in Fig. 10, the statistics view can be sliced and diced, and drilled down for further inspection.

Overall: Overall statistics for the whole cohort comprising all sections.

By Section: Statistics broken down by section.

By Question: Statistics by question, with option to drill down by section.

By Subquestion: Statistics by subquestion, again with option to drill down by section.

(5) Investigate: The last step in the Grader (labelled (5) in Fig. 1) has some heuristics to detect potential cheating efforts and inconsistencies (such as missing or wrong emails in student files as well as the similarities among the Answers and the Resources or the Solutions).

Grouped by	Mean	Std Dev	Median	Min	Max
All	8.62	0.99	8.95	3.25	12.00

Question	Mean	Std Dev	Median	Min	Max
1	8.58	0.95	8.95	3.25	9.00
3	0.04	0.26	0.00	0.00	3.00

Drill-down by Section

Section	Mean	Std Dev	Median	Min	Max
G3	8.69	0.78	8.95	5.50	10.00
G4	8.45	1.04	8.95	4.20	9.00
G5	8.73	0.97	8.98	5.50	12.00
G6	8.60	1.16	8.95	3.25	9.00

Question	Section	Mean	Std Dev	Median	Min	Max
1	G3	8.67	0.75	8.95	5.50	9.00
	G4	8.45	1.04	8.95	4.20	9.00
	G5	8.62	0.80	8.95	5.50	9.00
	G6	8.60	1.16	8.95	3.25	9.00
	G3					1.00

Fig. 10: Viewing statistics, showing the option to drill down.

Load Student List: Load the list of students for in-depth analysis for copying/cheating attempts, with buttons for further investigation, as shown in Fig. 11.

Build Database Tables: The cheating investigation is done by comparing each submitted file against all other files from all other students, as described in the following section. Because of the combinatorial computing requirement, we create an intermediate database table of similarities. This (one-time) process can be launched using this button. Although it will take a while, the process can be interrupted and restarted without losing what is already computed.

B. Investigating Academic Dishonesty

Once the student list is loaded in the Grader application for investigation, each student will appear in the list as shown in Fig. 11, near the top. The first button, **Prepare**, is to perform the pair-wise similarity scores (if needed). The second button, **Investigate**, performs the checks (4) and (5) in the list below, while the last button, **Summary**, performs checks (6) and (7). The heuristic rules that the Grader uses are as follows:

1) Compare the student's Answer file to the corresponding Resource file. If identical, the question was not attempted.
2) A large number of student Answers being identical or very similar to the Solution files may indicate a question leak.
3) Compare each student's Answer file to the corresponding file of all other students, and tabulate similarities.
4) In each Answer file, compare the email entered by the student to the one from which it is submitted. A mismatch indicates a potential attempt at cheating. (Fig. 11, left.)
5) For each *Answer file* from one student, find the corresponding file from other students with the largest similarity score. (Fig. 11, table on the left.)
6) For each *student*, find the number of times another *student* has the largest similarity score. A large frequency indicates potential collaboration among students. (Fig. 11, table on the right, middle.)
7) Compare the similarity score between the Answer files of one student with that of another student. When this similarity is high (while the similarity between the Answer and the Resource files is low, indicating that the student has worked on the file), it indicates cheating attempt also. (Fig. 11, table on the right, bottom.)

Note that the comparison is done after stripping the obvious variations (such as white spaces and the lines containing the student name and email). Quick exact comparisons are performed using one-way hash functions, while more nuanced comparison is delegated to PHP string comparison functions.

C. Pedagogical Objective

One hidden agenda we have in developing the Grader is to present it to the students as a real-world application that they

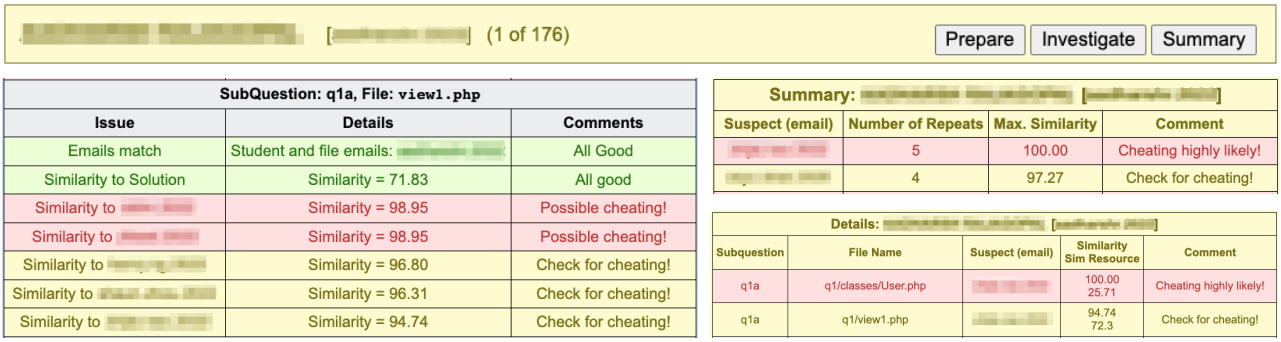

Fig. 11: Plagiarism detection.

can study and see the concepts they learn in the WAD courses in action. For this purpose, the Grader is developed using the same technology stack, namely WampServer and MAMP. We also use, to the extent possible, the conventions and paradigms to which the students were introduced.

However, since the WAD1 course is the first time most of our students come across object-oriented programming, they are not yet familiar with some of the advanced concepts like inheritance and polymorphism. In addition, some of the PHP-specific techniques we use in the development of the Grader, such as late static binding and dispatch handlers using magic methods, are not taught in the course. Furthermore, advanced web concepts such as CSS styling and browser interactivity with Javascript are also not taught in WAD1. Although we minimize their use in the Grader development, we need to employ some of it for usability. For instance, as seen in Figures 2, 3 and 8, we use syntax highlighting using a Javascript library. Because of these constraints, in the current iteration, we merely demonstrated the application in the classroom. We did not distribute it for their inspection. It is perhaps more appropriate to have them dissect it in the second course in the WAD series.

V. Experience and Future Directions

The Grader was used in the evaluation of two lab tests conducted for the WAD1 course during the spring term of 2023, involving a total of 176 students (Table I). The first question (**RQ1**) was regarding its utility in terms of reducing the instructor workload. Its implementation significantly reduced the grading workload and facilitated the efficient entry of marks into our e-learning system. Although we did not conduct a formal analysis to quantify the reduction in effort, the Grader's ability to effectively handle files, identify submission errors, track marks, and detect cheating attempts validated the development investment. Our estimate of the time saved, based on anecdotal evidence, is about 50% to 60%.

Our **RQ2** was on how effective the Grader application was, in terms of motivating and inspiring students. In order not to introduce a bias in our measurement, we did not pose a direct survey question to the students, which would probably result in most of the stating that they indeed found it inspiring. Instead, we relied on their unprompted comments about the Grader in their general comments about the instructor in their end-of-term feedback. Out of the 176 students who were exposed to the Grader, 169 provided feedback and about 10% of them mentioned the Grader, always in a positive light. Some of these comments are listed below.

- "...I find it even more inspirational because he then later [*sic*] shares with the class how he used PHP to automate grading with a live demo! It was very fascinating and piqued my interest."
- "...hence, the implementations Prof [*name removed*] has for his class enabled a much [*sic*] efficient learning environment..."
- "...he constantly uses his own examples of web design and application to show us how this can be applied in the real world contexts."
- "...I love his insights and sharing of best practices and his own practical use cases of whatever we learn. Gives me motivation whenever I feel like this subject is meaningless."
- Ïdots He also demonstrates how the things we learn in class can be put to use in real life by building certain websites with the concepts we were being taught."

From the unprompted comments in the student feedback, we conclude that the Grader served as a teaching tool, and consider **RQ2** answered in the affirmative.

The third question (**RQ3**), related to the efficacy of the Grader in detecting plagiarism could not be answered conclusively. Although the heuristic algorithm implemented to identify instances of academic dishonesty successfully flagged certain student responses for further investigation, among the 176 students assessed using the Grader, no evidence of cheating attempts was discovered upon closer manual inspection.

In our context, where lab tests were closely invigilated, the absence of cheating attempts was not unexpected. However, should we expand the use of the Grader to take-home assignments, which inherently offer more opportunities for academic dishonesty, we anticipate a higher likelihood of detecting such behaviors. Nevertheless, if students intent on engaging in plagiarism are given ample time, they may find ways to evade the Grader's detection mechanisms effortlessly. SThis could necessitate the implementation of more robust detection

algorithms like Moss, JPlag, or the Plagiarism Detection System (PDS) for assignments. In any case, based on the data collected from our deployment of the Grader, we cannot settle **RQ3**.

Going forward, we plan to expand the deployment of the Grader to additional sections of web application development courses, as well as other programming courses employing a similar workflow. Furthermore, we intend to leverage the Grader as a pedagogical tool in WAD courses, utilizing its capabilities to inspire and motivate students in their learning journey. We also plan to bolster the plagiarism and cheating detection using text analytics as well as bespoke techniques for computer languages and source code.

VI. Conclusion

In this article, we presented the Grader, a grading assistant application employed for assessing lab tests in our web application development course. Although numerous tools exist in literature for automated grading and plagiarism detection, we have yet to encounter one that streamlines the entire grading process, encompassing file management, score tracking, and providing summary statistics. The implementation of the Grader addresses this gap, providing a comprehensive system that significantly reduces the grading workload for instructors evaluating lab tests. It can be utilized in other programming courses that follow a similar workflow, including submission to an e-learning platform.

For web application development courses using the MAMP or WampServer stack, the Grader serves as a valuable pedagogical resource, offering students exposure to a substantial real-life project, encouraging the practical application of their acquired knowledge. Based on their feedback, students seem to appreciate this hands-on approach, finding it inspiring.

Moving forward, our future plans involve deploying the Grader to a larger cohort and exploring its utilization in formative assessments, such as assignments. While its plagiarism detection feature may require further refinements before using it for take-home assignments, its notable capabilities of robust file handling, automated grading, syntax highlighting etc. are certain to be of use. The Grader application is freely available from the authors upon request.

References

[1] P. Navrat and J. Tvarozek, "Online programming exercises for summative assessment in university courses," *ACM International Conference Proceeding Series*, vol. 883, pp. 341–348, 06 2014.

[2] E. Riese and O. Bälter, "A qualitative study of experienced course coordinators' perspectives on assessment in introductory programming courses for non-cs majors," *ACM Transactions on Computing Education*, vol. 22, 03 2022.

[3] M. Fabro, E. Almeida, and F. Sluzarski, "Teaching web application development: A case study in a computer science course," *Informatics in Education*, vol. 11, pp. 29–44, 04 2012.

[4] P. Alston, D. Walsh, and G. Westhead, "Uncovering threshold concepts in web development: An instructor perspective," *Trans. Comput. Educ.*, vol. 15, pp. 2:1–2:18, 03 2015.

[5] K.-B. Yue and W. Ding, "Design and evolution of an undergraduate course on web application development," in *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 22–26.

[6] J. Hage and P. Rademaker, "A comparison of plagiarism detection tools," 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:16448906

[7] T. H. Park and S. Wiedenbeck, "Learning web development: Challenges at an earlier stage of computing education," in *Proceedings of the Seventh International Workshop on Computing Education Research*, ser. ICER '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 125–132.

[8] Y. D. Wang and N. Zahadat, "Teaching web development in the web 2.0 era," in *Proceedings of the 10th ACM Conference on SIG-Information Technology Education*, ser. SIGITE '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 80–86.

[9] M. Qadah and S. M. Al-Shomrani, "Teaching web development course in information system department," in *2011 3rd International Congress on Engineering Education (ICEED)*, 2011, pp. 165–168.

[10] M. Multazam, Z. Syahrial, and R. Rusmono, "Development of learning models in web programming courses with computer-based learning tutorials," *Turkish Online Journal of Distance Education*, vol. 24, pp. 232–244, 04 2023.

[11] J. Paiva, J. Leal, and A. Figueira, "Automated assessment in computer science education: A state-of-the-art review," *ACM Transactions on Computing Education*, vol. 22, 02 2022.

[12] N. Strijbol, C. Petegem, R. Maertens, B. Sels, C. Scholliers, D. Peter, and B. Mesuere, "Tested—an educational testing framework with language-agnostic test suites for programming exercises," *SoftwareX*, vol. 22, p. 101404, 05 2023.

[13] T. Delev and D. Gjorgjevikj, "E-lab: Web based system for automatic assessment of programming problems," in *ICT Innovations 2012, Web Proceedings ISSN 1857-7288*, 2012, pp. 75–84.

[14] C. Cohenour and A. Hilterbran, "Automated grading of excel workbooks using matlab," in *Proceedings of the 2016 ASEE Annual Conference and Exposition*, 2016.

[15] B. A. Bertheussen, "Power to business professors: Automatic grading of problem-solving tasks in a spreadsheet," *Journal of Accounting Education*, vol. 32, no. 1, pp. 76–87, 2014.

[16] M. Thulasidas, "Secure answer book and automatic grading," in *2020 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, 12 2020, pp. 564–569.

[17] R. Weegar and P. Idestam-Almquist, "Reducing workload in short answer grading using machine learning," *International Journal of Artificial Intelligence in Education*, 02 2023.

[18] A. Krsak, "Curbing academic dishonesty in online courses," in *Proceedings of TCC 2007*. TCCHawaii, 2007, pp. 159–170.

[19] G. R. Cluskey, C. Ehlen, and M. Raiborn, "Thwarting online exam cheating without proctor supervision," *Journal of Academic and Business Ethics*, vol. 4, 01 2011.

[20] R. McHaney, T. Cronan, and D. Douglas, "Academic integrity: Information systems education perspective," *Journal of Information Systems Education*, vol. 27, pp. 153–158, 01 2016.

[21] C. Ko and C. Cheng, "Secure internet examination system based on video monitoring," *Internet Research*, vol. 14, pp. 48–61, 02 2004.

[22] L. Kong, Z. Han, H. Qi, and Z. LU, "A ranking-based text matching approach for plagiarism detection," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101.A, pp. 799–810, 05 2018.

[23] J. Vykopal, V. Švábenský, P. Seda, and P. Čeleda, "Preventing cheating in hands-on lab assignments," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1*, ser. SIGCSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 78–84.

[24] R. del Pino, E. R. Royo, and Z. J. H. Figueroa, "A virtual programming lab for moodle with automatic assessment and anti-plagiarism features," in *Proceedings of the 2012 International Conference on e-Learning e-Business Enterprise Information Systems & e-Government*, 2012.

[25] B. Kurniawan, A. Purnomo, I. Idris, K. Adi, and I. D. Eskasasnanda, "Using spada brightspace to enhance pedagogical skills in teacher professional program," *International Journal of Emerging Technologies in Learning (iJET)*, vol. 15, p. 180, 04 2020.