sustainability

MDPI

Article

Effect of an Instructor-Centered Tool for Automatic Assessment of Programming Assignments on Students' Perceptions and Performance

Aldo Gordillo

Departamento de Ingeniería de Sistemas Telemáticos, ETSI Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain; a.gordillo@upm.es

Received: 16 September 2019; Accepted: 8 October 2019; Published: 10 October 2019

Abstract: Automated assessment systems are increasingly used in higher education programming courses since the manual assessment of programming assignments is very time-consuming. Although a substantial amount of research work has been done on systems for the automatic assessment of programming assignments, most studies are focused on technical characteristics and further research is needed for examining the effect of using these systems on students' perceptions and performance. This paper examines the effect of using an instructor-centered tool for automatically assessing programming assignments on students' perceptions and performance in a web development course at a higher education institution. A total of three data sources were used: a survey to collect data regarding students' perceptions, and the grades of the student assignment submissions and of a practical programming exam in order to analyze the students' performance. The results show that the incorporation of the automated assessment tool into the course was beneficial for the students, since it allowed for increasing their motivation, improving the quality of their works, and enhancing their practical programming skills. Nonetheless, a significant percentage of students found the feedback that was generated by the tool hard to understand and of little use, and the generated grades unfair.

Keywords: automated assessment; automated feedback; computer science education; educational technology; programming; technology-enhanced learning

1. Introduction

Programming is one of the essential skills that students should gain through computer science and software engineering degrees that are offered by higher education institutions. However, many students find this skill difficult to learn, as evidenced by the high failure rates that programming courses usually have [1]. On most of these courses, student programming assignments are used for both learning and assessment. In European universities, the use of this type of assignments has notably increased over the last years due to the construction of the European Higher Education Area (EHEA), which required them to adapt their degrees and use new teaching methodologies that aimed to promote continuous assessment. This adaptation has posed serious problems in some countries, like Spain, since continuous student assessment is very difficult to put in practice in overcrowded engineering classes where there is a high student-teacher ratio since manual monitoring of students is unfeasible [2].

A wide range of learning benefits of using student programming assignments has been reported in the literature [3]. Programming assignments help students to understand how software development principles can be applied, to translate their theoretical knowledge into practical programming skills, and enhance these skills. From the teachers' point of view, student programming assignments provide information regarding how students have met the learning objectives of the course and their performance. The programming assignments need to be graded and effective feedback should be provided to the students in order to take advantage of the aforementioned benefits. However, the manual assessment of assignments is a complex, tedious, and very time-consuming task, even for small classes. For classes with a high student-teacher ratio, the manual assessment of student programming assignments becomes a difficult or even an impossible task. Universities can deal with this issue by increasing the number of teachers. However, this approach is not usually feasible due to economic reasons. Another option that universities have for dealing with this issue without increasing personnel costs is to use automated assessment systems.

Automated assessment systems can yield several benefits for both teachers and students [4–11]. Firstly, the use of these systems drastically reduces the workload of the teachers, thus allowing for continuous student assessment in programming courses, even for those with a high student-teacher ratio. It has been proved that these systems allow for teachers to give more programming assignments and, at the same time, spend less time on preparing and grading these assignments [12]. Secondly, using automated assessment systems can increase the consistency and objectivity of the assessment of programming assignments. This way, students get grades not dependent on personal considerations. Furthermore, some features of programming assignments might even be better assessed by automated approaches than by humans [5]. There is also evidence that suggests that student programming skills are better evaluated while using computer-based assessment rather than paper and pencil testing [13]. Finally, feedback automatically generated by assessment systems can provide students with information about the quality of their solutions and guidance on how to improve them. Feedback allows for students to become aware of their own difficulties and limitations. Even if the generated feedback is rudimentary, it can yield a notable improvement for the students' learning experience [14]. Thus, automated feedback allows for providing students with more personalized guidance than would otherwise be possible in large programming courses.

Although there is no doubt that using automated assessment systems in programming courses can have many advantages, it should be pointed out that the success of the use of an automated assessment system ultimately relies on how it is applied [15]. Therefore, these systems will only be beneficial for teachers and students if they are appropriately incorporated into a course. An incorrect application of an automated assessment system can even have negative effects on students' engagement and performance. For instance, an issue that is related to the educational implications of using automated assessment systems often raised in the literature is that students easily begin to rely on these systems to test their programs and do less testing on their own [4,5,14]. Taking all of these into account, it becomes clear that the effect of using an automated assessment system in a programming course strongly depends on the learning methodology that is used in the course. Evidence of this fact can be found in the literature. For example, García-Mateos and Fernández-Alemán [16] found that an automated assessment system considerably reduced the dropout rates in a programming course while Rubio-Sánchez et al. [17] evaluated the same system in a course with a different learning methodology and did not observe a statistically significant reduction in dropout rates. The authors of this latter study suggested that the different methodology was the key factor that explained the effect on the dropout rate. It is worth stressing that the use of automated assessment systems in programming courses has some important implications for teachers. Mainly, these systems require teachers to be more careful with the pedagogical design of the programming assignments [4,5,14,15]. Besides, the introduction of automated assessment systems could change how students approach these assignments [6], lead them to cheat [5,15], alter their behavior during classroom sessions [14], and generate grades that they perceive as unfair [18]. In conclusion, using automated assessment systems in programming courses can have both advantages and drawbacks, and hence teachers should carefully think about the implications of their use and the learning methodology to be used before incorporating one into a course.

Automated assessment systems can be used for both summative and formative assessment. On the one hand, they can be used for summative purposes in practical programming exams as well as in homework assignments. On the other hand, automated assessment systems can be used to show the students their assessment results (usually a grade and feedback) and allow them to resubmit their programming assignments. Thereby, students can use the automatically generated feedback to realize the mistakes that they have made, learn from them, and improve their works. To prevent students from using automated assessment systems as a formal testing mechanism instead of performing their own tests, these systems generally provide two technical solutions: allowing the teachers to limit the maximum number of submissions per assignment and to set a minimum waiting time between submissions. Thereby, students are forced to do testing on their own as well as to think more thoroughly about the quality of their works before submitting them for evaluation. Other solutions have been proposed in the literature, such as using compulsory time penalties, slightly modifying the assignment after each submission, and limiting the amount of feedback [6].

According to how the assessment process is triggered, automated assessment systems can be classified as instructor-centered (when the instructors start the assessment process), student-centered (when the students start the assessment process), and hybrid (when the system implements a strategy aimed at exploiting the strengths of both instructor-centered and student-centered approaches) [8]. This classification by approach reflects the balance between the provision of more immediate feedback to students and better control of the assessment by the instructors. In instructor-centered automated assessment systems, students do not receive immediate feedback but the teachers can strongly control the assessment process, compare students' solutions to detect plagiarism, and review the assessments before showing the feedback to the students. If the automated assessment system used is student-centered, the students can receive feedback immediately, but the teachers cannot review it before it is provided and have less control over the assessment process. When a hybrid approach is used, the assessment system provides students with partial feedback immediately after the submission and the teachers can strongly control part of the assessment process. The approach that is used by an automated assessment system to calculate the programming assignment grades is also an important issue, since there is evidence that different approaches can lead to significant differences in grades for some cases [19]. Besides, the feedback that is provided could be different depending on the assessment approach used. Assessing the functionality of students' code through batteries of tests is still the most commonly used approach to grade programs, but there are other grading techniques, such as semantic similarity-based grading [20]. Moreover, it should be taken into account that some grading approaches require assistance from instructors to calculate the grades. For instance, the Algo+ tool [21] requires an instructor to assign feedback and scores to some of the most frequent submissions.

Many automated assessment systems exist and are used nowadays. This fact is evidenced by several literature reviews that were conducted over the past years [4–11], in which more than 100 different systems to automatically assess programming assignments have been identified. The main reason for this high number of systems is that existing automated assessment systems are generally difficult to extend and adapt to particular requirements [14]. Although many research works have been published on automated assessment systems for programming assignments, most of them are focused on technical characteristics. Further research work is needed for examining the effect of using these systems on students' perceptions and performance in different educational settings and in combination with different learning methodologies. This research gap is especially pronounced for instructor-centered automated assessment systems, where the teachers instead of the students start the assessment process.

This paper examines the effect of using an instructor-centered tool for automatically assessing student programming assignments on students' perceptions and performance in a web development course at a higher education institution. The main contribution of this paper is insights on how the students perceived the instructor-centered automated assessment tool (overall acceptance, usefulness, motivation, and grading fairness), and on how this tool influenced their academic performance. The results of this paper help to have a better understanding of the benefits and drawbacks of using instructor-centered automated assessment systems in programming courses at a higher education level.

The rest of the paper is organized as follows. The next section reviews related work on systems for automatic assessment of student programming assignments, with emphasis on research addressing the effect of these systems on students' perceptions and/or performance. Section 3 describes the automated assessment system that was examined in this work, as well as the context in which it was used. Section 4 explains the evaluation methodology and Section 5 shows the results of the conducted evaluation. Finally, last section finishes with the conclusions of the paper and some directions for future work.

2. Related Work

Teachers of programming courses have relied on software systems for assessing programming assignments for more than 50 years [4]. Over the past years, several literature reviews have been conducted describing and classifying different automated assessment systems for programming assignments [4–11]. In 2005, Douce, Livingstone, and Orwell [4] reviewed a number of influential automated assessment systems and classified them according to their age. In this same year, Ala-Mutka [5] surveyed several automatic approaches for assessing programming assignments and identified the program features that they are able to assess, which include functionality, efficiency, testing skills, and coding style. Ihantola et al. [6] presented a systematic literature review of automatic assessment tools for programming assignments comprising the 2006–2010 period, in which a total of 17 tools were analyzed. This review aimed to investigate the features of these systems that were reported in the literature after 2005. In 2013, Caiza and Del Alamo [7] reviewed 11 tools for automatic grading of programming assignments. The features analyzed were programming languages supported, programming language used to develop the tool, logical architecture, deployment architecture, work mode, evaluation metrics, and technologies used. Three years later, in 2016, Souza, Felizardo, and Barbosa [8] conducted a systematic literature review through which they identified 30 assessment tools for programming assignments and classified these tools based on assessment type (manual, automatic, or semiautomatic), approach (instructor-centered, student-centered, or hybrid), and specialization (contests, quizzes, software testing, or non-specialized). This review of the literature also identified and discussed the main characteristics of these tools: electronic submission, automated checking, automated grading, immediate feedback, availability of a problem repository, submission history storage, and statistics reporting. In the same year, Keuning, Jeuring, and Heeren [9] published another literature review. In this work, a total of 69 tools that provide automated feedback on student solutions were identified and analyzed with a focus on their generated feedback. Recently, in 2018, Lajis et al. [10] published a review of techniques in automatic programming assessment, in which they identified several automated assessment systems. Lastly, another recent literature review was published by Ullah et al. [11] in 2018. This work reviewed 17 automated assessment systems and discussed their strengths and limitations in detail. The previous literature reviews have identified more than 100 automated assessment systems for programming assignments. Among the most popular of these systems, it is worth mentioning AutoLEP [22], Web-CAT [23], Mooshak [24], CourseMarker [25], BOSS [26], Marmoset [27], WebWork [28], Automata [29], INGInious [30], Coderunner [31], VPL [32], and HoGG [18].

Despite the wide adoption of automated assessment systems in programming courses at a higher education level, not enough research work has been done to examine the effect of their use on students' perceptions and performance. This section reviews some prior works from the literature that report on evaluations of automated assessment systems for student programming assignments in programming courses. Edwards [33] conducted an evaluation of a prototype of Web-CAT [23], which is an open source automated grading system for programming assignments that is capable of providing feedback both on the quality of the solutions and on the quality of their tests. The results of this evaluation show that students agreed that this prototype provided excellent support for test-driven development and they would like to use it in future classes. Although a comparison was made between the assignment grades that were achieved by the students of a course where test-driven development and Web-CAT were employed and those that were achieved by the students of another course in which a different automated assessment system was used, the actual effect of the tool on students' academic performance was not assessed. Several experiences from using Marmoset in several introductory programming courses for automatically assessing student assignments were described in [27]. The automated assessment system was found to be valuable for students, although its impact on student achievement was not assessed. García-Mateos and Fernández-Alemán [16] applied a new continuous evaluation methodology that was based on Mooshak [24] in a course on algorithms and data structures to reduce its dropout rate. Although the dropout rate of the course decreased, it is not clear whether this decrease was due to the use of Mooshak or to the methodology that was introduced at the same time. Another empirical evaluation of Mooshak was later performed by [17]. This evaluation concluded that there was no evidence to claim that the use of Mooshak helped to decrease the dropout rate, that the generated feedback needs to be richer to improve student acceptance, and that students with higher grades tend to solve assignments faster than their peers. A limitation of the aforementioned study is that the Mooshak perception was evaluated through a questionnaire with a sample of only 34 students. Furthermore, the effect on students' performance of introducing Mooshak in the course was not investigated. In [34], a Moodle module for automatically assessing VHDL-based assignments capable of providing feedback to students was presented. Students' opinions were collected through a survey, whose results show that students were satisfied overall. Nonetheless, the authors concluded that the feedback generated by the system needs to be improved in future versions because students had difficulties to understand it. This Moodle module was later extended to support the automatic verification of code written in Matlab [35]. The results of a survey conducted among students show that they were generally pleased with this new system, although the automatic feedback reports were found to be somewhat difficult to understand.

Another interesting related work is the one of Amelung, Krieger, and Rösner [14], in which the authors introduced a service-oriented approach for the development and deployment of flexible and reusable software components for automatic assessment of programming assignments. An automated assessment system that was developed using this approach was evaluated through questionnaires showing that it had positive effects on students' and teachers' perceptions. The effect of the system on students' performance was not addressed. Wang et al. [22] evaluated the impact of AutoLEP on students' performance in an introductory C programming course at a higher education institution with 120 students and found that students who used AutoLEP achieved slightly higher average final grades than those students who did not use this system. Notwithstanding, statistical data that allow for concluding that this difference was statistically significant were not provided. Questionnaires to measure students' perceptions of AutoLEP were conducted and, although detailed results were not reported, the authors qualitatively described the student feedback that was gathered, which was generally positive. Another relevant work is the one of Jurado, Redondo, and Ortega [36], who used an automated assessment system to build an intelligent tutoring system for learning programming. 27 students evaluated the implemented system through a survey after a lab session and the results show that, in general, these students were satisfied with it. Pieterse [15] presented a prototype of an instructor-centered automated assessment system and an experience report of the use of that system. The author also identified a set of factors that are likely to contribute to the successful application of automatic assessment of programming assignments. These factors include the quality of the assignments, clear formulation of tasks, well-chosen test data, good feedback, unlimited submissions, student testing maturity, and additional support.

De-La-Fuente-Valentín, Pardo, and Kloos [37] used an automated delivery and assessment system in a large programming course to manage and semi-automatically assess student submissions for two software projects that students developed following a PBL (Project Based Learning) methodology. The analysis of this case study concluded that a majority of students would like to use the system in other courses, that the use of the system allowed for students to improve their project scores, and that it helped to reduce the dropout rate. An online assessment system to automatically grade programming assignments was presented by Bai et al. [12]. Based on empirical data, the study showed that this system allowed instructors to save time and, at the same time, assign more homework and provide quicker feedback. The quality of students' interaction with the assessment system and the impact of its feedback on students' performance were not analyzed. Another recent related work is the one of Bakar et al. [38], which reports on the development and evaluation of an automated assessment system. This research work is quite limited, as the evaluation merely consisted of a questionnaire that was answered by 11 lecturers and former students that only addressed perceived usability and perceived effectiveness. Finally, Restrepo-Calle, Ramírez, and González [39] examined the interaction between the students and a student-centered automated assessment tool called DOMjudge [40] through quantitative analysis, as well as the students' perceptions toward this tool through interviews. The authors found that, on average, students received two pieces of feedback from the assessment tool before finding the correct solution. They also found a positive correlation between final course grades and the percentage of correct solutions submitted to the assessment tool among the total number of attempts. However, no comparison was made between students using the system and students not using it. Based on the results of the interviews, the authors concluded that students believed that DOMjudge had great potential as an assessing and learning tool and that it was a valuable source of feedback, although some students mentioned that this feedback should be broader in terms of explaining or suggesting corrections for the program assessed.

Despite the research that has been carried out so far in the field, more work is needed to better understand the benefits and challenges of using instructor-centered automated assessment systems in computer programming courses. This paper advances the state of the art in this area by examining the effect of using an instructor-centered tool for automatically assessing student programming assignments both on students' perceptions and their academic performance. Novel contributions of this work include a performance comparison of students who used feedback that was provided by an instructor-centered automated assessment system with those who did not, a measure of effectiveness of single instances of automated feedback, and new insights of how students perceive instructor-centered automated assessment systems, including findings regarding how fair students feel the grades generated by these systems are.

3. Description of the Case Study

3.1. Course Context

This work examines the effect of using an instructor-centered tool for automatic assessment of student programming assignments on students' perceptions and performance in a web development course at a higher education institution. This course is a core course for third-year students and it is part of the Bachelor's Degree in Telecommunications Engineering from UPM (Universidad Politécnica de Madrid). It lasts one semester and it is worth 4.5 ECTS (European Credit Transfer System) credits, equivalent to approximately 115–135 hours of student work. The course introduces students to web development covering the HTML, CSS, and JavaScript programming languages, as well as the Node.js JavaScript runtime environment. The course methodology is based on lectures that are given by the teachers (students have three class hours per week), which are complemented with videos available on a Moodle platform. The course is divided into two parts: the first part covers HTML, CSS, JavaScript, and an introduction to Node.js, and the second part focuses on web development with Node.js. Each part is separately evaluated, has the same duration and weight in the final grade (50%), and it has to be passed with a grade greater than or equal to 5 out of 10 to pass the course.

The main reason for introducing the automated assessment tool in the course was that its high student-teacher ratio made it impossible to manually assess all of the student programming assignments that a continuous assessment approach requires. This need arose with the construction of the European Higher Education Area (EHEA), which required European universities to adapt their degrees in order to, among other purposes, promote continuous assessment. Nevertheless, the course staff considered it to be appropriate to gradually incorporate it, since it was the first year in which the assessment tool was used in the course, and hence it was used only in the first part.

During the first part of the course, students had to turn in a total of four programming assignments. Three of these assignments consisted of developing client-side web applications by using HTML, CSS, and JavaScript, and the fourth assignment required students to develop a Node.js application. For each assignment, a detailed description of the application to be developed was provided, including a careful specification of its requirements using natural language. The assignments were delivered to students through the Moodle platform used in the course, and students submitted their applications to this same Moodle platform through single files by using common formats (ZIP, RAR, TAR, ...) or through URLs pointing to a public web server. The grades of the programming assignments did not count toward the final grade of the course, but the students were warned that completing them was important to prepare for the course exams.

Students had two opportunities to submit each of the four assignments. First, they had a deadline of one week to deliver the first submission. Once this deadline was reached, a teacher used the automated assessment tool that was addressed in this study to generate a grade and an instance of feedback for each submission. After reviewing the automatic assessment, the teacher published the grades and feedback of the assignment on the Moodle platform of the course. Details regarding the operation of the assessment tool are provided in the next section. Students received the feedback and grades of their first submission one to three days after its deadline, depending on the assessed assignment. Once students received feedback on the quality of their first submission and guidance on how to improve it, they had the possibility to review and improve their application and resubmit it to get a new grade and a new piece of feedback. Students were given one week for delivering this second submission. Subsequently, the teacher used the assessment tool again for assessing the new submissions and published the new grades and feedback on the Moodle platform of the course. This publication took place one to three days after the deadline of the second submission. It is worth pointing out that, although students were encouraged to turn in all programming assignments to prepare for the course exams, it was not mandatory for them to improve and resubmit their solutions. Therefore, there was a group of students who did not use the feedback that was provided by the assessment tool and another group who used this feedback to improve their applications and learn from their mistakes. It is also important to point out that limiting the maximum number of submissions per assignment to two in this course prevented students misusing the automated assessment tool as a trial and error device, an issue that has been often raised in the literature [4,5,14]. Limiting the number of possible resubmissions forced students to test their own applications and to more thoroughly think about design and testing issues as well as about the quality of their solutions before submitting them for evaluation.

At the end of the first part of the course, the students took a practical programming exam in a computer lab. This practical exam consisted of two equally weighted exercises: one HTML and CSS exercise, and one JavaScript exercise. These exercises aimed to assess the essential practical programming skills that students should have gained through the first part of the web development course.

3.2. *Automated Assessment Tool*

This section provides an overview of the automated assessment system used in the web development course described above to automatically grade the programming assignments that were submitted by the students and to provide them with timely feedback.

Although a wide range of automated assessment systems existed [4–11], the course staff had to develop a custom assessment tool because no system that met all of the requirements was found. This was not an unusual case since, unfortunately, instructors and educational institutions often have to develop their own assessment systems instead of reusing existing ones. One of the main reasons for this seems to be that existing automated assessment systems for programming assignments are generally difficult to extend and to adapt to particular requirements [14]. The automated assessment tool developed was called IAPAGS, which stands for Instructor-centered Automated Programming Assignments Grading System. This tool was designed in such a way that it meets the following requirements:

- IAPAGS is instructor-centered and its use is transparent for students. These features allowed for incorporating the automated assessment tool into the course without modifying the course methodology or the course syllabus.
- IAPAGS can automatically generate feedback and grades for student programming assignments uploaded through Moodle 2.6 (the version of Moodle used in the course at the time of the study) without modifying the Moodle instance.
- IAPAGS is able to assess client-side web applications that students can submit through a single file by using common formats (ZIP, RAR, TAR, ...) or through a URL pointing to a public web server.
- IAPAGS is able to assess Node.js applications submitted through single files using common formats.
- IAPAGS allows for teachers to define their own test cases and to specify the generated feedback as well as how the grades of the assignments are calculated.
- When using IAPAGS, it is possible to execute the students' source code in a secure environment.

The fact that the IAPAGS tool is instructor-centered implies that the course teachers can start and control the assessment process of the student programming assignments. Before showing the results of an assessment (grade and feedback) to the students, the teachers can review the automatic assessment, and verify the generated grades and feedback. Thereby, if they found an error on the automatic assessment or if they realize a possible improvement, they can modify the tests of the assignment and generate a new set of grades and feedback. Another advantage of the instructor-centered approach is that, since it requires limiting the maximum number of submissions per assignment, it avoids the risk of students misusing the assessment tool on a trial and error basis. Thus, as explained before, it forces students to test their applications and carefully think about their quality before submitting them for evaluation. Furthermore, the use of an instructor-centered automated assessment tool also allows for the course teachers to compare students' submissions in order to verify their uniqueness. In this case study, no plagiarism detection software was used because the assignment grades did not count toward the final course grade, and thus students had no reason to cheat. The major shortcoming of the instructor-centered approach is that students do not receive immediate feedback, since they have to wait for the teachers to verify the results of the automatic assessment. In this case study, the teachers published the results of the automatic assessment on the Moodle platform one to three days after the submission deadline, depending on the assessed assignment.

Another important characteristic of the IAPAGS tool is that it is completely transparent for students. Although it is incorporated into a course, students do not need to be aware of whether their assignments are going to be manually or automatically assessed, they just have to worry about submitting an application that meets all of the requirements that were specified by the problem statement of the assignment. The course teachers could even decide whether to assess a particular assignment manually, automatically, or semi-automatically after students have submitted their works. In the case study that was analyzed in this paper, the students were told that their assignments were going to be assessed using an automated assessment tool, but that the teachers would verify the automatic assessment. It is also worth indicating that IAPAGS can be used for grading programming assignments, but also for grading programming exams.

Figure 1 represents the workflow for teachers and students when IAPAGS is used in a course for assessing a student programming assignment in order to better illustrate how the IAPAGS tool works. This workflow is comprised of the following 10 steps:

1) First, the course teachers create the programming assignment for the students. On the one hand, the problem statement of the assignment must be elaborated, describing the requirements of a software application that students will have to develop in detail. On the other hand, the teachers

must also develop a battery of tests for the IAPAGS tool that will be used to automatically assess the software applications that are submitted by the students. Details about how these tests have to be implemented are provided later in this section. Besides the problem statement and the battery of tests, it is also recommended that the teachers produce a solution for the assignment and test it against its battery of tests.

2) The teachers publish the problem statement of the assignment on the Moodle platform of the course. The battery of tests can remain private.
3) The students download the problem statement of the assignment. They are given a certain period of time to develop the software application described in this statement and submit that application to the Moodle platform.
4) The students develop the requested software application. For this task, the students are free to use any development tool (code text editor, IDE-Integrated Development Environment, debugging tool, ...) that they want.
5) The students submit the application they have developed for the assignment to the Moodle platform.
6) Once the assignment deadline is reached, the teachers download all of the software applications that were submitted by the students in the previous step to the Moodle platform.
7) The course teachers use the IAPAGS tool to generate a grade and an instance of feedback for each student submission. The battery of tests developed for the assignment in the first step is used in order to conduct this automatic assessment with IAPAGS. The output of the automatic assessment that is conducted by IAPAGS consists of two files: a text file containing a report with details about the assessment (errors occurred during the process, grades and feedback for each submission, the result of each individual test and the score given by these tests, messages for instructors, paths of the files and URLs assessed, ...), and a Moodle compliant CSV file that contains all of the grades and feedback for the assessed submissions.
8) The teachers review the results of the automatic assessment performed by the IAPAGS tool. At this point, they can fix errors or make improvements in the assessment (e.g., by providing more detailed feedback for certain issues or by generating more accurate grades). To do this, the teachers should modify the assignment's battery of tests and then perform a new automatic assessment. In this phase of the assessment process, the teachers also have the option of comparing students' submissions to detect plagiarism.
9) After verifying the results of the automatic assessment, the course teachers import the CSV file that was generated by the IAPAGS tool (in step 7) to the Moodle platform to add the generated grades and feedback to the course gradebook, which makes this information available to the students.
10) The students can see the results of the automatic assessment: the grade of their assignment and feedback on their submission. Based on this feedback, the students can improve their application and, if the course teachers allow it, resubmit it to get new feedback and a better grade.

The architecture of the IAPAGS assessment tool is comprised of three main components, as shown in Figure 2: the assignment manager, the submission manager, and the assignment test suite file. Teachers interact with the IAPAGS tool while using a command-line interface. The assignment manager processes the instructions that are given by the teachers via this interface. This component is also in charge of retrieving all of the student assignment submissions from the file system and delivers them, one by one, to the submission manager for their assessment. When the assignment manager invokes the submission manager, it processes and verifies the submission. If the submission file needs to be uncompressed, parsed in order to extract a URL or treated in some other way, this task is performed by the submission manager. After verifying a submission, this component runs the test suite of the assignment (defined in the assignment test suite file) against this submission by using Mocha [41], which is an open source JavaScript test framework that runs both on Node.js and in the browser. The submission manager monitors the execution of the tests defined by the assignment test suite and generates an assessment report based on the results of such tests. This report contains detailed information on the conducted assessment, including the identifier of the submission in Moodle and the student name (obtained from the name of the submission file), the identifier of the assignment in Moodle (obtained from the assignment test suite file), errors that occurred during the assessment, path of the file or URL assessed, the grade for the submission on a 0–100 scale, feedback for the student, and messages for the instructors. Besides, this report includes, for each test run, its result (pass or fail) and the score with which it contributes to the assignment grade.

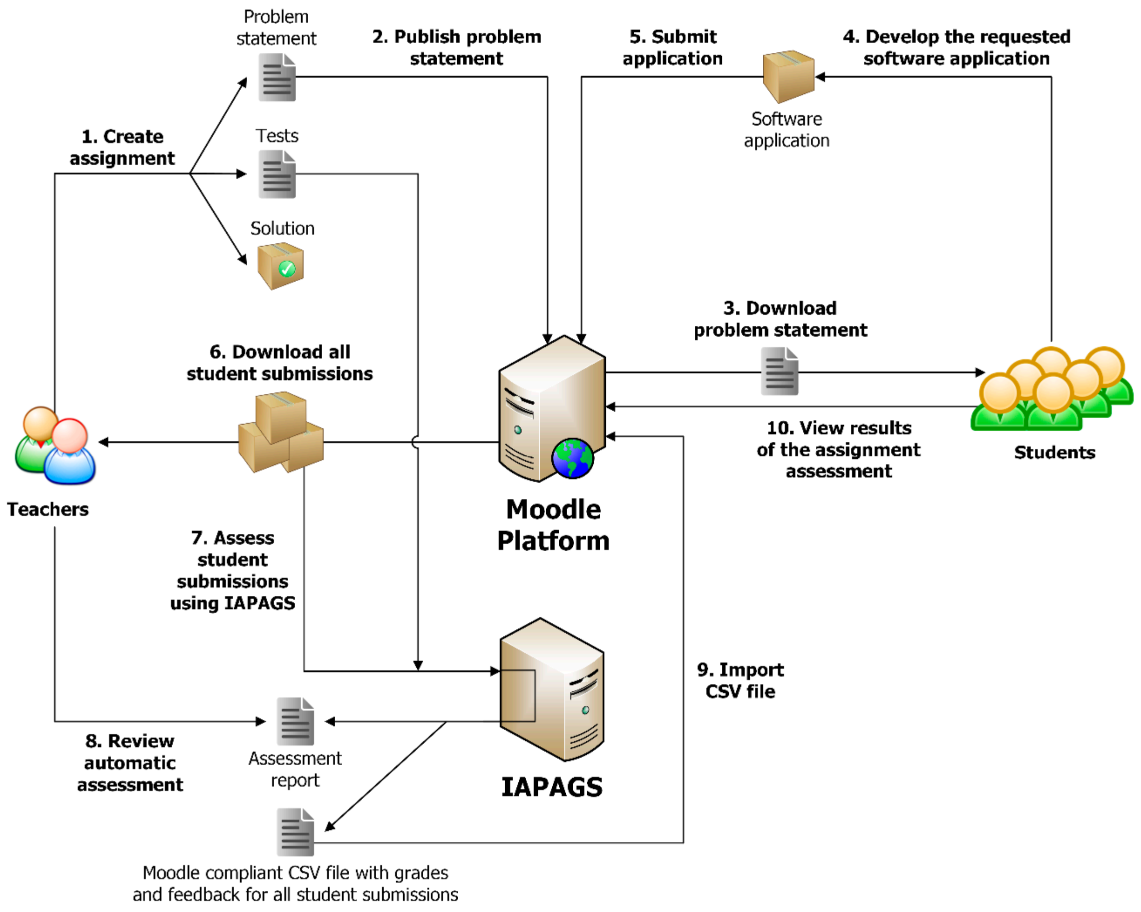
Problem statement
2. Publish problem statement
5. Submit application
4. Develop the requested software application
Software application
1. Create assignment
Tests
Solution
3. Download problem statement
6. Download all student submissions
10. View results of the assignment assessment
Students
Teachers
Moodle Platform
7. Assess student submissions using IAPAGS
9. Import CSV file
8. Review automatic assessment
Assessment report
IAPAGS
Moodle compliant CSV file with grades and feedback for all student submissions

Figure 1. Workflow of the Instructor-centered Automated Programming Assignments Grading System (IAPAGS) tool.

The assignment manager and the submission manager are part of the core software of the IAPAGS tool. However, the assignment test suite file must be produced by the course teachers for each assignment, as mentioned in the first step of the IAPAGS workflow. This file should define a collection of tests for assessing the student submissions of a specific assignment. As IAPAGS uses the Mocha JavaScript testing framework [41] for conducting the automatic assessment, these tests should be developed for this framework. The assignment test suite files that teachers must create for automatically assessing the programming assignments with the IAPAGS tool have the same structure as a file defining a standard test suite for Mocha. The only difference is that teachers should use a specific syntax for naming the test suite to indicate the identifier of the assignment in Moodle, as well as for naming the different tests in order to define their assigned score (i.e., their weight), feedback for students in case of success, feedback for students in case of failure, warning messages for students, messages for instructors, and error messages. Therefore, teachers only need to know how to use the Mocha test framework and this syntax to develop their own batteries of tests for the IAPAGS tool. Teachers with knowledge of Mocha can start using IAPAGS almost immediately, given that the syntax is very straightforward. It is also worth mentioning that Mocha allows for using different assertion libraries for writing the tests, including Chai [42], the assertion library used in the case study that was reported in this paper.

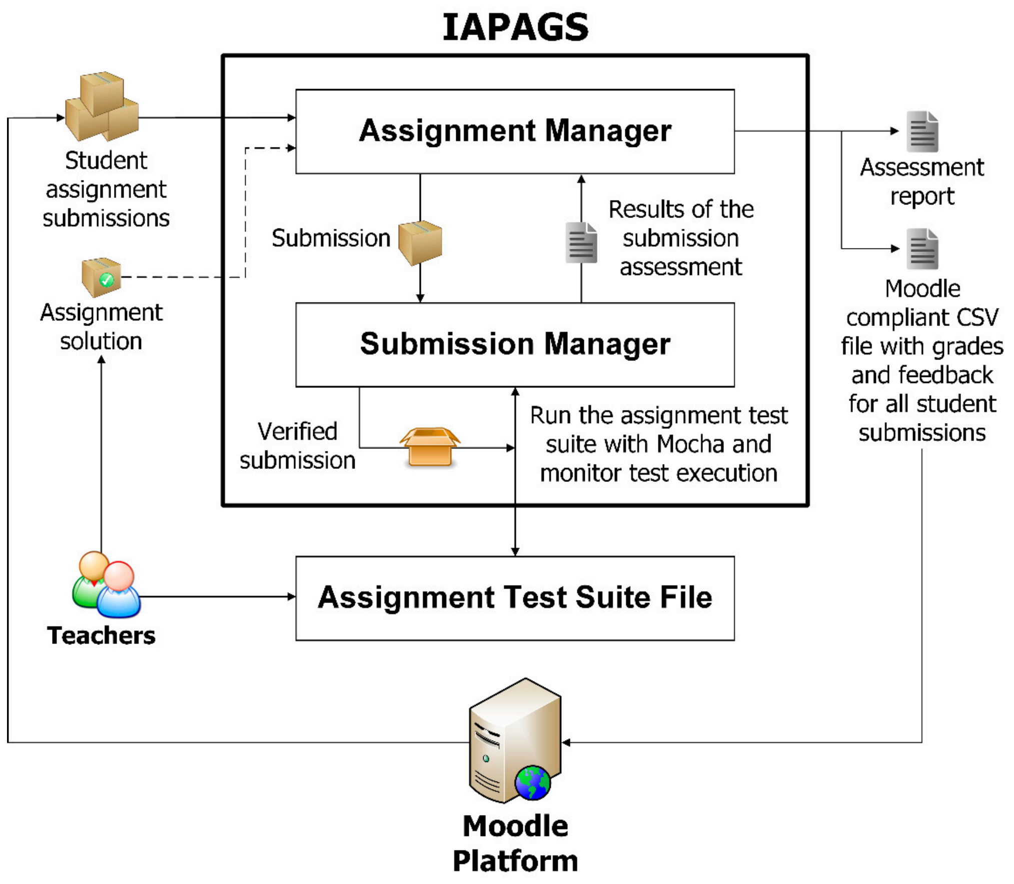
IAPAGS
Assignment Manager
Student assignment submissions
Submission
Results of the submission assessment
Assignment solution
Submission Manager
Verified submission
Run the assignment test suite with Mocha and monitor test execution
Assessment report
Moodle compliant CSV file with grades and feedback for all student submissions
Teachers
Assignment Test Suite File
Moodle Platform

Figure 2. Architecture of the IAPAGS tool.

Once the submission manager has generated a report with the results of a submission assessment based on the execution of the tests defined in the assignment test suite file, this report is returned to the assignment manager. The assignment manager stores all of the results that were provided by the submission manager, and continues to invoke it until there are no more submissions to assess. IAPAGS allows for setting a time limit for testing submissions in such a way that, if the submission manager exceeds this time to assess a submission, the assignment manager aborts the assessment and generates a report for this submission that includes a timeout error. This time limit is an essential feature for automated assessment systems, because the code of the applications that are submitted by the students can contain infinite loops or infinite recursions. In this regard, Restrepo-Calle, Ramírez, and González [39] found that nearly 12% of the errors made by the students that were identified by an automated assessment system were because the program under evaluation exceeded the maximum run-time that was allowed. Once all of the submissions have been assessed, the assignment manager generates a final assessment report by aggregating the results of all submission assessments. Teachers can read this report to review the results of the automatic assessment that was conducted by IAPAGS. Additionally, the assignment manager generates a Moodle compliant CSV file containing all the grades and feedback for the assessed submissions. The aim of this file is to enable teachers to easily add these grades and feedback to the course gradebook in Moodle. Therefore, the output of an automatic assignment assessment conducted by the IAPAGS tool consists of a report with detailed information regarding the performed assessment, including messages for the course teachers, and a file for allowing these teachers to import to Moodle the grades and feedback generated by IAPAGS for the assessed student assignment submissions.

When teachers create an assignment to be automatically assessed with IAPAGS, they draft the problem statement, create a test suite, and, although it is not required by the IAPAGS tool, they also generally develop a solution for the assignment. Teachers can use the IAPAGS tool to test this solution against the assignment test suite and check that the generated grade is the maximum and the generated feedback is as expected in order to verify the correctness of the assignment solution (as well as of the tests).

It is worth noting that IAPAGS needs to run the applications that are developed by the students in order to automatically grade them, which is a risky task, since these applications might have malicious code or bugs that lead them to attempt actions that could cause damage to the computer in which the tool is executed. Therefore, in order to execute the students' source code without security risks, teachers should use the IAPAGS tool from a secure environment (or sandbox environment).

Regarding privacy, it should be clarified that IAPAGS does not permanently store any data. The files of the student assignment submissions that are downloaded by the teachers from Moodle can be deleted after conducting the automatic assessment with IAPAGS. Similarly, the assessment report and the Moodle compliant CSV file containing all of the grades and feedback for the assessed submissions can be deleted after reviewing the results of the automatic assessment and importing the generated grades and feedback to the course gradebook. Therefore, all private data are only stored permanently on the Moodle platform of the course.

Finally, it is important to mention some limitations of the IAPAGS automated assessment tool. The current version of IAPAGS only supports Moodle 2.6, although it would be easy to modify the tool to support other learning management systems, as long as they allow downloading student assignment submissions and importing grades and feedback. Another relevant limitation of IAPAGS is that it can only be used to automatically grade software applications that can be tested by using the Mocha JavaScript testing framework. An additional limitation of IAPAGS, which is a common limitation of all automated assessment systems that use test-based grading approaches (i.e., systems that automatically assess the functionality of students' code through batteries of tests), is that when the assessment system is unable to successfully execute or interpret a software application submitted by a student due to errors in the source code (and thus is also unable of running the tests), the programming assignment is graded with a zero and the feedback that is provided to the student is generic. This same behavior occurs when the software application under evaluation exceeds the maximum run-time that is allowed.

4. Evaluation Methodology

The objective of the evaluation was to analyze the effect of using the IAPAGS instructor-centered automated assessment tool on students' perceptions and performance in the web development course previously described.

Data regarding the students' perceptions toward the IAPAGS tool were collected through an online survey, which was administered to all students at the end of the first part of the course after publishing the exam grades. Participation in the survey was anonymous and voluntary in order to garner the most honest responses. The survey included two initial demographic questions (age and gender), a closed question about how difficult students find learning programming, a list of statements with which students needed to agree or disagree using a five-point Likert scale, and an open-ended question asking students for suggestions and additional comments. The survey included elements addressing the key aspects of the automated assessment tool: overall opinion, usefulness of the generated feedback, usefulness and fairness of the generated grades, quickness, motivation, and suitability of the system to be used in similar courses. The survey sample was composed of a total of 94 students.

Two data sources were used in addition to the student survey to evaluate the effect of IAPAGS on the students' performance: the grades of the student assignment submissions and the grades of the practical programming exam. On the one hand, the grades of the student assignment submissions allowed for examining to what extent the feedback that was provided by the automated assessment tool was useful for students to improve their initial solutions. On the other hand, the grades of the practical programming exam allowed for determining whether those students who used the feedback generated by the automated assessment tool outperformed their peers. A particular characteristic of this study is that the instructor-centered automated assessment tool was incorporated into the course in such a way that students only had two opportunities to submit each of their four programming assignments: one before receiving any feedback and another one after receiving feedback on their first submission automatically generated by the tool. The grades of student assignment submissions were used to perform a comparison between the grades of the first submissions and those of their corresponding resubmissions. This comparison allowed for measuring the effect of single instances of feedback generated by IAPAGS on students' performance, since it compared the grades of the assignment submissions before and after the students made use of such feedback. With the purpose of obtaining a reliable measure of this effect, only those pairs of student assignment submissions for which both a first submission and a resubmission existed were used for the comparison analysis. Thereby, this analysis involved 348 student assignment submissions (76 for the first assignment, 110 for the second one, 126 for the third, and 36 for the fourth), half of which corresponded to first submissions and the other half to resubmissions. Lastly, comparing the exam grades that were achieved by the students who used the feedback that was generated by the tool with the grades of those who did not use it was also used to assess the effect of the IAPAGS instructor-centered automated assessment tool on students' performance. A total of 240 students took the exam, of which 111 (46%) had used the feedback generated by the automated assessment tool during the course.

5. Results and Discussion

5.1. Student Survey

The sample of the student survey consisted of 94 students, 69 males (73%) and 25 females (27%), 20–29 years of age (M = 20.9, SD = 1.3). With regard to the question about the difficulty of learning programming, around half of the students (53%) stated that they did not find programming an especially difficult subject, whereas 12% stated the opposite and the remaining 35% neither agreed nor disagreed with that statement. The results of the survey are shown in Table 1, including, for each question, the mean (M) and standard deviation (SD).

The overall opinion of the IAPAGS automated assessment tool recorded a mean of 3.1 on a 1–5 scale and the students clearly answered that they prefer to have the reports generated by the assessment tool rather than not have feedback at all (M = 4.3, SD = 1.0). Furthermore, the results indicate that using the IAPAGS tool in the course allowed for increasing the students' motivation to complete their programming assignments (M = 3.7, SD = 1.2). The fact that students' motivation increases when they know that their submissions are actually going to be reviewed was also observed by [11,14]. Taking into account that, in previous years, no feedback or grades could be provided to the students on their programming assignments due to the high student-teacher ratio of the course, the results clearly show that the incorporation of the automated assessment tool into the course was very beneficial. A similar conclusion was reached by Restrepo-Calle, Ramírez, and González [39] through quantitative analyses, who determined that DOMjudge [40] offered a more comprehensive feedback to the students of a programming course than the one they would have received using just a compiler. Consistent findings were also published by [34,35], who surveyed students of previous courses and found that they agreed that the inclusion of an automated assessment system notably enhanced the learning experience.

Although the mean overall opinion was higher than the Likert scale midpoint, it is worth pointing out that nearly one of every three students had a negative opinion about the IAPAGS tool. One of the main reasons for this is that a significant percentage of students found the feedback generated by the tool hard to understand (44%) and not very useful (39%). These results are consistent with those that were reported in some previous studies that evaluated the use of other systems for automatically assessing student programming assignments. For instance, García-Mateos and Fernández-Alemán [16] claimed that the feedback that was provided by Mooshak [24] should be improved in the future and [17] reported that students found Mooshak's feedback poor after conducting a survey. In the work carried out by Gutiérrez et al. [34], although a majority of students surveyed found the feedback provided by an automated assessment system somewhat useful, the authors concluded that this feedback needs to be improved, because students experienced difficulties to understand it. Similarly, Ramos et al. [35] conducted a student survey and reported that students found the automatic feedback reports generated by an assessment system somewhat complicated to understand. In another previous study [39], some interviewed students complained about the richness of the feedback that was provided by DOMjudge. Somewhat contradictory findings that are related to automated feedback usefulness have also been reported in the literature. For example, Spacco et al. [27] carried out a student survey and reported that 67% of students who had used Marmoset in a programming course stated that they were able to make good use of feedback provided by this automated assessment system, whereas a minority (24%) disagreed with this statement. In the study conducted by Jurado, Redondo and Ortega [36], 67% of the students who had used an automated assessment system during a programming lab session indicated, through a survey, that the explanations that were provided by the system agreed with the delivered solutions, and 89% thought that the test cases were useful.

Table 1. Results of the student survey on the IAPAGS tool (N = 94).

Question	M	SD
Please, state your level of agreement with the following statements: (1 Strongly disagree–5 Strongly agree)		
My overall opinion of the automated assessment tool is positive.	3.1	1.2
I prefer the reports (i.e., feedback and grades) generated by the automated assessment tool rather than not receiving any feedback at all	4.3	1.0
I complete the programming assignments with greater motivation if I know that I am going to receive feedback and a grade generated by the automated assessment tool	3.7	1.2
The reports generated by the automated assessment tool were easily understandable	2.7	1.2
The reports generated by the automated assessment tool were useful	3.0	1.3
I have improved my software applications thanks to the reports generated by the automated assessment tool	3.0	1.2
The feedback generated by the automated assessment tool allowed me to discover bugs in my software applications that otherwise would have remained unnoticed	3.0	1.3
I believe that the reports generated by the automated assessment tool were provided quickly	3.0	1.2
The grades generated by the automated assessment tool were fair	2.5	1.2
I would like this type of automated assessment system to be used in other programming subjects	3.2	1.4

A reason why some students found the feedback of little use and hard to understand in this study is because the code of some of the applications they submitted for assessment contained infinite recursions or infinite loops, and in these cases the feedback that was generated by the automated assessment tool used in the course consisted just in a generic error message, without clearly indicating the source of the error. Future versions of the tool should be able to provide meaningful specific feedback when these errors occur. Another reason was the fact that students used, for their own testing, a different web browser than the one used by the automated assessment tool. Ideally, all the HTML5 features should work in the same way in all HTML5 compliant web browsers. However, the reality is that the support of this standard slightly differs in the different web browsers and hence a web application might behave distinctly depending on the web browser used. Thus, in some cases, the web applications that were submitted by the students worked correctly on their web browsers, but produced errors when evaluated by the assessment tool. A finding that was similar to the previous one was reported by Rubio-Sánchez et al. [17], who pointed out that the major source of dissatisfaction with Mooshak's feedback was the fact that students used a different compiler so that, although their programs worked correctly on their computers, they produced errors on Mooshak's server. In the case study addressed in this paper, some students even submitted applications whose code contained errors that were not raised by their web browsers (for being laxer about code structure), but that prevented the web browser used by the automated assessment tool to load such applications. In these cases, given the impossibility of running the tests, the feedback provided to the students was also generic. A possible solution for future years would be to force students to perform the testing with a specific web browser instead of allowing them to choose it freely, as the different HTML5 compliant web browsers do not operate in the same way. It should be noted that the reasons exposed in this paragraph are supported by the comments students made in the open-ended question. In general, in these comments, students expressed that the feedback generated by the automated assessment tool should be more extensive, more accurate, and better explain how to fix the reported errors.

Regarding the usefulness of the generated feedback, it is also worth mentioning that, in this study, around 36% of the surveyed students stated that the automated assessment tool allowed for them to improve the software applications that were developed for the assignments, as well as to discover bugs in such applications that otherwise would have remained unnoticed. In this regard, Gutiérrez et al. and Ramos et al. [34,35] also provided evidence by means of student surveys that automated assessment systems can help students to detect bugs. Moreover, Ramos et al. [35] found that, like in this study, students were, on average, neutral to the statement claiming that the automated feedback was useful for amending their works. In conclusion, the results that were obtained in the student survey related to automated feedback usefulness strongly evidence that the feedback provided by the IAPAGS tool on the programming assignments brought several benefits to the students, but it has significant room for improvement.

When inquiring about the time that they had to wait before accessing the assessment results of the programming assignments, students neither agreed nor disagreed with the statement that these results were provided quickly (M = 3.0, SD = 1.2). In the comments section of the survey, several students suggested immediately providing the feedback, as well as increasing the number of allowed submissions (some students expressed that they would like to have unlimited submissions) in order to have more opportunities to correct the errors that were detected by the tool. The main disadvantage of instructor-centered automated assessment systems with respect to other types of automated assessment systems is the lack of immediate feedback after each submission. The results of this survey suggest that, although students clearly prefer instant feedback, instructor-centered automated assessment systems can also be used to provide them with timely feedback. With respect to the other request of the students, although it is obvious that they do not like to have a limited number of allowed submissions for each programming assignment, this limitation is necessary for pedagogical reasons, since otherwise there is a risk of students misusing the assessment tool as a trial and error system. This incorrect application of automated assessment systems, which has been noticed by previous research works [4,5,14], should be prevented, since it can have negative effects on students' learning.

Regarding the grades for the programming assignments that were generated by the automated assessment tool, more than half of the students (54%) thought that these grades were unfair, whereas only a 26% perceived such grades as fair (the remaining 20% were neutral about grading fairness). In the open-ended question of the survey, there were some complaints about the low grades that the automated assessment tool generated in some cases. There was also a complaint that the automatic assessment approach did not take into account the effort that was invested by the students in order to implement additional features in the applications. Criticisms on this issue were expected, because, as already indicated by Douce et al [4], a disadvantage of automated assessment systems is that they cannot award additional marks for creative design or innovative solutions. In view of these results, it becomes clear that, besides the perceived quality of the feedback, another reason why a considerable percentage of the students had a negative opinion on the automated assessment tool was that they thought that the generated grades were unfair. The main reason for which a majority of students perceived the grades as unfair was because they made certain errors that caused the automated assessment tool to misgrade the actual value of their effort. For example, when the students misspelled the name of a JavaScript function in a programming assignment, although this error could be considered small and not indicative of the programming skills for which they are being evaluated, the automated assessment tool graded the assignments very severely. Therefore, in these cases, students probably achieved lower grades than if the assignments had been manually assessed by a teacher. Other errors that were made by the students that caused the assessment tool to generate unfair grades in some cases were the inclusion of infinite recursions or infinite loops, as well as errors in the HTML code that prevented the web applications to be loaded by the web browser that was used by the assessment tool for not being fully compliant with the HTML5 standard. In these cases, besides not receiving accurate feedback (as explained above), the students were graded with a zero because the assessment tool was unable to successfully execute the tests against the submitted application. Naturally, in these cases, the students were strongly disappointed with the provided assessment results. Morris [18] had already observed that there is a number of errors that students can make that, although minor, can cause any automated grading system to be more severe than intended. He also developed a categorization of these errors, which was composed by the following six categories: naming errors, output format errors, indirect dependency, direct dependency, infinite loops, and infinite recursion. The findings reported in this study constitute a novel contribution to the field, as no work was found in the literature reporting empirical data on students' perceptions regarding grading fairness of automated assessment systems for programming assignments.

When asked about the possibility of using an automated assessment system similar to IAPAGS in other programming subjects, a majority of students (68%) were neutral or agreed with that possibility. Given that, according to the survey results, the students clearly prefer to be provided with feedback automatically generated by IAPAGS rather than not being provided with any feedback, the main reason why a notable number of students disagree with this possibility seems to be the negative perceptions of grading fairness. A measure that could be adopted to alleviate these negative perceptions is to perform an additional verification of the applications immediately after their submission, and allow for the students to repeat such submissions when the assessment tool is unable to execute or interpret the submitted application. Thereby, cases where students do not receive accurate feedback or are graded with a zero for this reason would be eradicated. Another measure that could be very helpful is to provide students with strong guidance regarding the use of the automated assessment tool at the beginning of the course, which encourages them to pay special attention to aspects such as naming, infinite loops, debugging, and testing.

In general, in the comments section of the survey, students stated that they strongly believe that the provided assessment results were much better than nothing. Furthermore, they praised the potential of the automated assessment tool that was used in the course, although they expressed that this tool should be improved for future years, especially in terms of grading fairness as well as the accuracy and readability of the automated feedback. A few students complained about the programming assignments being too demanding and several suggested resolving the programming assignments in class after the submission deadlines as well as to do more practical exercises during the lessons. This fact suggests that, although automated assessment systems can drastically reduce the teachers' workload by avoiding manual assessment, there is still a need for teachers to guide students toward successfully solving programming assignments.

Finally, perceptions of students who find programming especially difficult to learn were compared with those of students who do not by means of a Student's t-test. Regarding this point, it is worth highlighting that students who claimed to have less difficulties in learning programming had a better overall opinion on the automated assessment tool (M = 3.2, SD = 1.3 vs. M = 2.7, SD = 1.1), and found the reports that were generated by it more useful (M = 3.1, SD = 1.2 vs. M = 2.7, SD = 1.2) and less hard to understand (M = 2.8, SD = 1.2 vs. M = 2.3, SD = 0.9). However, none of these differences was found to be statistically significant at the *p*-value < 0.05 level, according to the Student's t-test.

5.2. Student Assignment Submissions

As explained in the evaluation methodology section, in this case study the students were only allowed two submissions per each of the four programming assignments. A different period was enabled for each of these submissions. The first submissions were performed before providing any feedback, whereas for the second submissions students had the possibility to improve their works based on the feedback that was received on their first submissions generated by the automated assessment tool that was employed in the course. In some cases, students made the two submissions allowed per assignment, while students made just one submission in other cases, either during the period enabled for the first submissions or during that enabled for the second ones. Table 2 summarizes all of the student assignment submissions received, including the grades that were achieved by the students on a 0–100 scale. First submissions refer to the submissions that were made by the students during the first period, while second submissions encompass all of those performed during the second period. The term "resubmissions" refers to those second submissions that were performed by students who had also performed a first submission. The final student assignment grades were calculated as the maximum between the grades achieved in the first and second submission.

A total of 1055 assignment submissions were received from 240 students. Of these submissions, 700 corresponded to the first period and the remaining 355 to the second period. Of the 355 second submissions, 174 (49%) were resubmissions and the other 181 (51%) corresponded to students who made a single assignment submission during the period enabled for the second submissions. A relevant fact is that, in only 25% of the cases where students made a first submission, they also made a resubmission, that is, a second submission to provide an improved version of the software application included in the first submission. This percentage rises to just 39% when only first submissions graded below 50 (out of 100) are taken into account. This figure indicates that, in more than half of the cases, students whose first submission was graded poorly decided not to amend their submitted work (i.e., in more than half of the cases, students did not perform a resubmission in order to achieve a better grade in spite of having a very low grade in the programming assignment). According to the comments that were left by the students in the open-ended question of the survey, the reason for this was a lack of time. This issue was especially notorious for the fourth assignment, whose submission periods overlapped with a considerable number of homework assignments and exams of other courses resulting in just 18 students submitting a solution during the period for first submissions, and 177 doing so during the second period.

Table 2. Student assignment submissions.

Assignment	Students Who Submitted the Assignment			First Submissions			Second Submissions		
	N	Grade		N	Grade		N (Resubmissions)	Grade	
		M	SD		M	SD		M	SD
#1	236	81	28	230	74	34	44 (38)	80	30
#2	230	62	35	229	55	36	57 (55)	67	34
#3	235	52	28	223	46	27	77 (63)	57	32
#4	177	31	35	18	49	37	177 (18)	31	35

Regarding grades, a clear trend of decrease between the first and last programming assignment can be seen. The average grade for the first assignment was 81 (on a 0–100 scale), while the average grades for the second, third, and fourth assignments were 62, 52, and 31, respectively. A slight decrease in grades was expected, since programming assignments were designed to be of increasing difficulty.

However, the remarkably low average grade that was achieved by the students in the fourth assignment was an unexpected and disheartening finding. The main reason for this finding is probably that students dedicated less time to this assignment, because, as mentioned before, during the period intended for its accomplishment they decided to focus on studying for exams and completing homework for other courses.

A comparison was made between the grades of the resubmissions performed by the students after making use of the automated feedback and the grades of the corresponding first submissions in order to measure the effect of the single instances of feedback generated by the automated assessment tool used during the course on the students' performance. Therefore, only those first submissions for which a resubmission was made were used for the comparison analysis, whose results are shown in Table 3.

Table 3. Comparison between grades of first student assignment submissions and their corresponding resubmissions.

Assignment	N	First Submissions		Resubmissions		Paired Aamples t-test p-Value (2-tailed)	Cohen's d Effect Size
		Grade		Grade			
		M	SD	M	SD		
#1	38	40	43	80	29	< 0.001	0.97
#2	55	38	32	68	33	< 0.001	0.89
#3	63	40	27	66	28	< 0.001	0.96
#4	18	49	37	63	30	0.03	0.57

The number of resubmissions for the first, second, third, and fourth assignment was, respectively, 38, 55, 63, and 18. A paired samples t-test was conducted in order to determine whether there was a significant difference between the grades that were achieved by the students in the first submissions and those achieved by them in the corresponding resubmissions. Furthermore, Cohen's d effect size was used to determine the practical significance of the difference in grades. When Cohen's d is used as a measure of effect size, a value of 0.2 indicates a small, 0.5 a medium, and over 0.8 a large effect size [43]. The results of the comparison analysis show that the grades achieved by the students in the resubmissions were statistically significantly higher (p-value < 0.05) than those that they achieved when they submitted the assignment for the first time. The grade increase for the four assignments was, respectively, 40%, 30%, 26%, and 14% (27.5% on average). Large effect sizes (Cohen's d > 0.8) were found for assignments one to three, whereas a medium effect size (Cohen's d = 0.57) was found for the fourth assignment. Based on these results, it can be suggested that the automated assessment tool that was used in the course clearly had a significant positive effect on the grades of student programming assignments. This finding was expected since the feedback that was provided by the tool informed students about their mistakes and suggested them how to enhance the software applications that they submitted.

Taking into account that some first submissions were graded with a zero due to programming errors that prevented the assessment system to execute or interpret the software application, it would be reasonable to think that the previously reported effect size could be inflated. However, given that less than 10% (17 out of 174) of the first submissions used for the comparison analysis received a zero due to the assessment system being unable to successfully execute the tests, it can be stated that the reported effect size is a reliable measure of the practical significance of the difference in grades between resubmissions and their corresponding first submissions.

It is worth pointing out that, although the results of the comparison presented in this section prove that students who made resubmissions were able to significantly improve their initial solutions based on a single instance of feedback provided by the assessment tool, the surveyed students were, on average, neutral to the statement that was included in the survey that claims that the automated assessment tool was useful for improving their software applications. According to the data regarding the student assignment submissions received, an explanation for this apparent contradiction is that, although the provided feedback was generally useful enough to significantly improve the student works, many students lacked the necessary time to amend and enhance their software applications based on such feedback. This explanation is also consistent with the students' comments complaining about the programming assignments being too demanding and the lack of time. Nevertheless, it should also be considered that, in a few cases, students might not have made a new submission, not because of lack of time, but rather because they did not find the feedback useful for improving the evaluated application.

The findings that were reported in this section are consistent with those of [37], who observed an improvement in the scores that were achieved by the students in the programming assignments with respect to the previous year, in which no automated assessment system had been used in the course. In another related work [39], the percentage of solutions considered to be correct by an automated assessment system out of the total number of programs submitted by the students was found to be around 33% (i.e., approximately one out of three submitted programs was assessed as correct). In the case study reported in this paper where the IAPAGS tool was used, considering as correct those submissions that scored greater than or equal to 50/100, the ratio of correct submissions to total submissions was around 55%. Nonetheless, if only solutions graded above 90/100 (which do not have any error or have minor errors) are considered as correct, this ratio decreases to 26%.

5.3. Exam Grades

A comparison was made between the exam grades achieved by the students who used the feedback generated by the tool and the grades of those who did not use it in order to analyze the effect of the automated assessment tool used in the course on the students' performance. The sample group of students who used the automated feedback was comprised of all those students who took the exam and that had used the feedback generated by the assessment tool at least once in order to make a resubmission of any of the course programming assignments. Table 4 shows the results of the comparison analysis.

Table 4. Grades achieved by the students in the practical programming exam.

Students Who Did not Use the Automated Feedback (N = 129)		Students Who Used the Automated Feedback (N = 111)		Independent Samples t-test p-Value (2-tailed)	Cohen's d Effect Size
M	**SD**	**M**	**SD**		
5.1	2.1	5.9	2.1	0.003	0.4

A total of 240 students took the exam, of which 111 (46%) had used the feedback that was generated by the automated assessment tool during the course and 129 (54%) had not. An independent samples t-test was conducted to determine if there was a significant difference in the grades achieved between the two groups of students. The magnitude of the difference in grades was determined by using Cohen's d effect size [43]. The results show that the students who used the feedback generated by the automated assessment tool throughout the course achieved better grades in the practical programming exam than those who did not. The difference between grades was found to be statistically significant (p-value < 0.05) with a small to medium effect size (Cohen's $d = 0.40$). Taking into account that students who used the automated feedback significantly outperformed those who did not in terms of practical programming skills, it can be suggested that the automated assessment tool used in the course had a positive effect on students' performance. This finding is consistent with [22], who found that students who used a system for the automatic assessment of programming assignments achieved somewhat higher average grades than their peers. In the previously cited work, the difference between student grades was found to be around 6%, which is a value that is similar to the one found in this study where the grades of the students who used the automated feedback were 8% higher. In this regard, it is worth mentioning that, unlike [22], this study provides statistical data that allow for stating that the difference in grades reported is statistically significant. In order to determine if there was a relationship between the assignment grades generated by the automated assessment tool used in the course and the grades achieved by the students in the practical programming exam, a Pearson correlation analysis was performed. This analysis revealed a statistically significant positive linear relationship (Pearson's r = 0.45, p-value < 0.001) between these two variables. A related finding was reported by Restrepo-Calle, Ramírez, and González [39], who found a statistically significant positive linear relationship (Correlation coefficient = 0.53, p-value < 0.001) between the percentage of solutions that were submitted by the students that were considered to be correct by an automated assessment system and the final course grade. In this case study, a statistically significant positive linear relationship was found between the previous percentage (considering as correct all student submissions graded above 90/100) and the grades that students obtained in the practical programming exam (Pearson's r = 0.31, p-value < 0.001).

6. Conclusions and Future Work

This paper analyses the effect of using an instructor-centered tool for the automatic assessment of programming assignments on students' perceptions and their performance in a web development course in a higher education setting. Three data sources were used for this study: a student survey, the grades of the student assignment submissions, and the grades that were achieved by the students in a practical programming exam. The reported results show that the incorporation of the automated assessment tool into the course was very beneficial for the students, since it allowed not only to provide them with automated feedback on their submissions, but also to increase their motivation and improve the quality of their software applications. Another major finding of this paper is that the students who used the feedback generated by the automated assessment tool throughout the course significantly outperformed those who did not in terms of practical programming skills. Nevertheless, a significant percentage of students found the feedback that was generated by the assessment tool difficult to understand and of little use. Furthermore, more than half of the students perceived the grades generated by the assessment tool as unfair. Therefore, there is a clear need for improving the tool in terms of grading fairness as well as the readability and accuracy of the feedback. In summary, this paper provides a significant contribution for the better understanding of the benefits and drawbacks of using instructor-centered automated assessment systems in higher education programming courses.

According to the survey data, students, on average, neither agreed nor disagreed with the statements that the feedback that was generated by the automated assessment tool was useful and allowed for them to improve their software applications. However, the data that were obtained on the assessment of the student assignment submissions and on the exam grades demonstrate that the students who used this feedback significantly improved their works and outperformed those who did not in terms of practical programming skills. Based on the obtained results, an explanation for this apparent contradiction is that, although the provided feedback was useful in most cases, many students lacked the necessary time to amend and enhance their software applications based on it. This fact points out that providing high-quality feedback does not guarantee an automated assessment system to be effective, because there may be other factors (e.g., high student workload) that hinder the use of that feedback.

The survey results that are presented in this study show that students prefer instant feedback and unlimited submissions. However, this approach is negative from a pedagogical perspective due to the risk of students relying on the automated assessment system to test their software applications and do less testing on their own [4,5,14]. Therefore, although students might have a less positive perception of instructor-centered automated assessment systems (in which the assessment process is started by the instructors) than of student-centered automated assessment systems, the effects of using the former on students' learning might be better if the latter are integrated without adopting effective measures to avoid system misuse. In this regard, future works could examine the instructional effectiveness of the different approaches to prevent the misuse of systems for automatic assessment of programming assignments. This work shows that, although students have a preference for immediate feedback, instructor-centered automated assessment systems are able to successfully provide them with feedback in a timely manner. This work also shows that providing students with a single instance of feedback for each programming assignment can significantly improve the quality of the submitted works as well as the students' programming skills.

Based on the evidence that is provided by this study, it can be suggested that automated assessment systems, when appropriately integrated into programming courses, can generate a significant improvement in students' performance. Students can enhance their programming skills if they carefully read the feedback that is generated by automated assessment systems, reflect on their mistakes, and try to improve their works after that. In this regard, it should be remarked that a single instance of automated feedback could allow students to significantly increase the quality of their works. The major problems of using automated assessment systems in programming courses seem to be the provision of poor feedback and the generation of unfair grades in a significant percentage of cases. The findings of this study, as well as those of several previous studies [16,17,34,35,39], indicate that, in general, the feedback that is generated by automated assessment systems for programming assignments should be richer, accurate, and easier to understand. Regarding the fairness of grades generated by automated assessment systems, it should be remarked that, although this topic has been previously addressed in the literature [18], no article was found examining the students' perceptions toward programming assignment grades generated by automated assessment systems. Thus, this paper makes a novel contribution by providing evidence on this topic. Taking the aforementioned facts into account, it can be concluded that the major drawbacks of using automated assessment systems for programming assignments over manual assessment is the generation of poor feedback and unfair grades in a significant percentage of cases.

There are several factors that teachers should take into account to successfully integrate and use an automated assessment system for programming assignments in a programming course. Firstly, the use of the assessment system should not introduce additional difficulties. Thus, either the use of the system is transparent for the students or such a system is easy to use and sufficient time is dedicated for students to learn how to use it. In this regard, it should be taken into account that the former approach is only possible by using instructor-centered automated assessment systems. Secondly, measures to force students to perform their own tests and think carefully about the quality of their solutions before submitting them for assessment should be adopted. The most frequent measures include a limitation of the maximum number of submissions per assignment and the establishment of a minimum waiting time between submissions. However, other measures can be adopted, including the use of compulsory time penalties, limitation of the amount of feedback provided, and modification of the assignments after each submission [6]. There are some measures that can be adopted to alleviate the problems that are related to poor feedback and unfair grades. Teachers should clearly indicate to students which environment is going to be used by the automated assessment system for assessing the programming assignments, including information related to the operating system, compiler, programming language, web browser, database management system, or any other software involved. Teachers can also provide students with software containers in order to avoid problems related to the environment. Thereby, the software developed by the students will work in the same way both in their computers and in the environment of the assessment system. Taking into account that there is a number of errors that students can make that, although minor, can cause automated assessment systems to be more severe in grading than intended [18], a measure that teachers can use to improve students' perceptions toward the grades generated by these systems is to clearly explain to the students how the assessment system used in the course works, and encourage them to pay special attention to these types of errors. Lastly, teachers should also take into account other factors that are likely to contribute to the successful application of automated assessment systems for programming assignments, which include, among others, quality and specification of the assignments, well-chosen test data, and additional support (e.g., provision of a request tracker or an online forum) [15]. Furthermore, in view of the results of this case study, the time that is required by the students to complete the programming assignments is also an important success factor. If the students' workload is too heavy, they might lack the time needed to make an effective use of the feedback provided.

A particular characteristic of this work is that the automated assessment system under study was used in a web development course for grading client-side web applications and Node.js applications. The experience reported in this paper constitutes another original contribution to the existing body of knowledge, as no work has been found in the literature reporting this use of an automated assessment system.

This article analyzes the effect of introducing an automated assessment system for programming assignments into a course in which it is not possible to provide manual feedback. Although this is a common scenario, there are many programming courses in which it is possible to provide manual feedback. Therefore, future work comparing the effect of using automated feedback versus manual feedback in different educational settings would be a valuable contribution. Furthermore, given that the effect of using a system for automatic assessment of programming assignments in a course strongly depends on the learning methodology that is used in the course, a valuable direction for future work would be to examine the effect of using the same automated assessment system with different learning methodologies. In this regard, it would also be interesting to study under which circumstances the use of an instructor-centered automated assessment system could be preferable to the use of a student-centered one and vice versa. Another possible future work could be to analyze the students' perceptions toward programming assignment grades generated by automated assessment systems that use different grading techniques, such as test-based grading and semantic similarity-based grading. Finally, although several reviews of the literature on software systems for assessing programming assignments have been carried out over the past years [4–11], a new systematic literature review on this topic with a focus on students' perceptions and performance would be a great contribution to the field.

Funding: This research received no external funding.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Watson, C.; Li, F.W.B. Failure rates in introductory programming revisited. In Proceedings of the 19th conference on Innovation and Technology in Computer Science Education (ITiCSE'14), Uppsala, Sweden, 21–25 June 2014; pp. 39–44.
2. Mora, M.C.; Sancho-Bru, J.L.; Iserte, J.; Sánchez, F. An e-assessment approach for evaluation in engineering overcrowded groups. *Comput. Educ.* **2012**, *59*, 732–740. [CrossRef]
3. Robins, A.; Rountree, J.; Rountree, N. Learning and teaching programming: A review and discussion. *Comput. Sci. Educ.* **2003**, *13*, 137–172. [CrossRef]
4. Douce, C.; Livingstone, D.; Orwell, J. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.* **2005**, *5*, 4. [CrossRef]
5. Ala-Mutka, K.M. A survey of automated assessment approaches for programming assignments. *Comput. Sci. Educ.* **2005**, *15*, 83–102. [CrossRef]
6. Ihantola, P.; Ahoniemi, T.; Karavirta, V.; Seppälä, O. Review of recent systems for automatic assessment of programming assignments. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling'10), Koli, Finland, 28–31 October 2010; pp. 86–93.
7. Caiza, J.C.; Del Alamo, J.M. Programming assignments automatic grading: review of tools and implementations. In Proceedings of the 7th International Technology, Education and Development Conference (INTED 2013), Valencia, Spain, 4–5 March 2013; pp. 5691–5700.
8. Souza, D.M.; Felizardo, K.R.; Barbosa, E.F. A systematic literature review of assessment tools for programming assignments. In Proceedings of the 2016 IEEE 29th Conference on Software Engineering Education and Training (CSEET 2016), Dallas, TX, USA, 5–6 April 2016; pp. 147–156.

9. Keuning, H.; Jeuring, J.; Heeren, B. Towards a systematic review of automated feedback generation for programming exercises. In Proceedings of the 21th conference on Innovation and Technology in Computer Science Education (ITiCSE'16), Arequipa, Peru, 11–13 July 2016; pp. 41–46.
10. Lajis, A.; Baharudin, S.A.; Kadir, D.A.; Ralim, N.M.; Nasir, H.M.; Aziz, N.A. A review of techniques in automatic programming assessment for practical skill test. *J. Telecommun. Electron. Comput. Eng.* **2018**, *10*, 109–113.
11. Ullah, Z.; Lajis, A.; Jamjoom, M.; Altalhi, A.; Al-Ghamdi, A.; Saleem, F. The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. *Comput. Appl. Eng. Educ.* **2018**, *26*, 2328–2341. [CrossRef]
12. Bai, X.; Ola, A.; Akkaladevi, S.; Cui, Y. Enhancing the learning process in programming courses through an automated feedback and assignment management system. *Issues Inf. Syst.* **2016**, *17*, 165–175.
13. Kalogeropoulos, N.; Tzigounakis, I.; Pavlatou, E.A.; Boudouvis, A.G. Computer-based assessment of student performance in programing courses. *Comput. Appl. Eng. Educ.* **2013**, *21*, 671–683. [CrossRef]
14. Amelung, M.; Krieger, K.; Rösner, D. E-assessment as a service. *IEEE Trans. Learn. Technol.* **2011**, *4*, 162–174. [CrossRef]
15. Pieterse, V. Automated assessment of programming assignments. In Proceedings of the 3rd Computer Science Education Research Conference (CSERC'13), Arnhem, Netherlands, 4–5 April 2013; pp. 45–56.
16. García-Mateos, G.; Fernández-Alemán, J.L. A course on algorithms and data structures using on-line judging. In Proceedings of the 14th conference on Innovation and Technology in Computer Science Education (ITiCSE'09), Paris, France, 6–9 July 2009; pp. 45–49.
17. Rubio-Sánchez, M.; Kinnunen, P.; Pareja-Flores, C.; Velázquez-Iturbide, Á. Student perception and usage of an automated programming assessment tool. *Comput. Hum. Behav.* **2014**, *31*, 453–460. [CrossRef]
18. Morris, D.S. Automatic grading of student's programming assignments: an interactive process and suite of programs. In Proceedings of the 33rd Frontiers in Education Conference (FIE 2003), Westminster, CO, USA, 5–8 November 2003; pp. 3–6.
19. Bey, A.; Jermann, P.; Dillenbourg, P. A comparison between two automatic assessment approaches for programming: An empirical study on MOOCs. *Educ. Technol. Soc.* **2018**, *21*, 259–272.
20. Wang, T.; Su, X.; Wang, Y.; Ma, P. Semantic similarity-based grading of student programs. *Inf. Softw. Technol.* **2007**, *49*, 99–107. [CrossRef]
21. Bey, A.; Bensebaa, T. ALGO+, an assessment tool for algorithmic competencies. In Proceedings of the 2011 IEEE Global Engineering Education Conference (EDUCON 2011), Amman, Jordan, 4–6 April 2011; pp. 941–946.
22. Wang, T.; Su, X.; Ma, P.; Wang, Y.; Wang, K. Ability-training-oriented automated assessment in introductory programming course. *Comput. Educ.* **2011**, *56*, 220–226. [CrossRef]
23. Edwards, S.H.; Pérez-Quiñones, M.A. Web-CAT: automatically grading programming assignments. In Proceedings of the 13th conference on Innovation and Technology in Computer Science Education (ITiCSE'08), Madrid, Spain, 30 June–2 July 2008; p. 328.
24. Leal, J.P.; Silva, F. Mooshak: A web-based multi-site programming contest system. *Softw. Pract. Exp.* **2003**, *33*, 567–581. [CrossRef]
25. Higgins, C.A.; Gray, G.; Symeonidis, P.; Tsintsifas, A. Automated assessment and experiences of teaching programming. *J. Educ. Resour. Comput.* **2005**, *5*, 5. [CrossRef]
26. Joy, M.; Griffiths, N.; Boyatt, R. The boss online submission and assessment system. *J. Educ. Resour. Comput.* **2005**, *5*, 2. [CrossRef]
27. Spacco, J.; Hovemeyer, D.; Pugh, W.; Emad, F.; Hollingsworth, J.K.; Padua-Perez, N. Experiences with Marmoset: designing and using an advanced submission and testing system for programming courses. In Proceedings of the 11th conference on Innovation and Technology in Computer Science Education (ITiCSE'06), Bologna, Italy, 26–28 June 2006; pp. 13–17.
28. Gotel, O.; Scharff, C. Adapting an open-source web-based assessment system for the automated assessment of programming problems. In Proceedings of the sixth IASTED International Conference on Web-based Education (WBE 2007), Chamonix, France, 14–16 March 2007; pp. 437–442.
29. Srikant, S.; Aggarwal, V. A system to grade computer programming skills using machine learning. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge Discovery and Data mining (KDD'14), New York, NY, USA, 24–27 August 2014; pp. 1887–1896.

30. Derval, G.; Gego, A.; Reinbold, P.; Frantzen, B.; Van Roy, P. Automatic grading of programming exercises in a MOOC using the INGInious platform. In Proceedings of the European MOOC Stakeholder Summit 2015 (EMOOCS'15), Mons, Belgium, 18–20 May 2015; pp. 86–91.
31. Lobb, R.; Harlow, J. Coderunner: A tool for assessing computer programming skills. *ACM Inroads* **2016**, *7*, 47–51. [CrossRef]
32. Rodríguez-del-Pino, J.C.; Rubio-Royo, E.; Hernández-Figueroa, Z.J. 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, Las Vegas, NV, USA, 16–19 July 2012.
33. Edwards, S.H. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.* **2003**, *3*, 1. [CrossRef]
34. Gutiérrez, E.; Trenas, M.A.; Ramos, J.; Corbera, F.; Romero, S. A new Moodle module supporting automatic verification of VHDL-based assignments. *Comput. Educ.* **2010**, *54*, 562–577. [CrossRef]
35. Ramos, J.; Trenas, M.A.; Gutiérrez, E.; Romero, S. E-assessment of Matlab assignments in Moodle: Application to an introductory programming course for engineers. *Comput. Appl. Eng. Educ.* **2013**, *21*, 728–736. [CrossRef]
36. Jurado, F.; Redondo, M.; Ortega, M. eLearning standards and automatic assessment in a distributed eclipse based environment for learning computer programming. *Comput. Appl. Eng. Educ.* **2014**, *22*, 774–787. [CrossRef]
37. De-La-Fuente-Valentín, L.; Pardo, A.; Kloos, C.D. Addressing drop-out and sustained effort issues with large practical groups using an automated delivery and assessment system. *Comput. Educ.* **2013**, *61*, 33–42. [CrossRef]
38. Bakar, M.A.; Esa, M.I.; Jailani, N.; Mukhtar, M.; Latih, R.; Zin, A.M. Auto-marking system: A support tool for learning of programming. *Int. J. Adv. Sci. Eng. Inf. Technol.* **2018**, *8*, 1313–1320. [CrossRef]
39. Restrepo-Calle, F.; Ramírez, J.J.; González, F.A. Continuous assessment in a computer programming course supported by a software tool. *Comput. Appl. Eng. Educ.* **2019**, *27*, 80–89. [CrossRef]
40. Eldering, J.; Gerritsen, N.; Johnson, K.; Kinkhorst, T.; Werth, T. DOMjudge. Available online: https://www.domjudge.org (accessed on 9 October 2019).
41. Mocha. Available online: https://mochajs.org (accessed on 9 October 2019).
42. Chai. Available online: https://www.chaijs.com (accessed on 9 October 2019).
43. Cohen, J. A power primer. *Psychol. Bull.* **1992**, *112*, 155–159. [CrossRef] [PubMed]