sustainability

MDPI

Article

Automated Assessment in Programming Courses: A Case Study during the COVID-19 Era

Enrique Barra [1,*], Sonsoles López-Pernas [1], Álvaro Alonso [1], Juan Fernando Sánchez-Rada [1], Aldo Gordillo [2] and Juan Quemada [1]

[1] Departamento de Ingeniería de Sistemas Telemáticos, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain; sonsoles.lopez.pernas@upm.es (S.L.-P.); alvaro.alonso@upm.es (Á.A.); jf.sanchez@upm.es (J.F.S.-R.); juan.quemada@upm.es (J.Q.)

[2] Departamento de Sistemas Informáticos, Escuela Técnica Superior de Ingenieros de Sistemas Informáticos, Universidad Politécnica de Madrid, 28031 Madrid, Spain; a.gordillo@upm.es

* Correspondence: enrique.barra@upm.es

Received: 5 August 2020; Accepted: 7 September 2020; Published: 10 September 2020

Abstract: The COVID-19 pandemic imposed in many countries, in the short term, the interruption of face-to-face teaching activities and, in the medium term, the existence of a 'new normal', in which teaching methods should be able to switch from face-to-face to remote overnight. However, this flexibility can pose a great difficulty, especially in the assessment of practical courses with a high student–teacher ratio, in which the assessment tools or methods used in face-to-face learning are not ready to be adopted within a fully online environment. This article presents a case study describing the transformation of the assessment method of a programming course in higher education to a fully online format during the COVID-19 pandemic, by means of an automated student-centered assessment tool. To evaluate the new assessment method, we studied students' interactions with the tool, as well as students' perceptions, which were measured with two different surveys: one for the programming assignments and one for the final exam. The results show that the students' perceptions of the assessment tool were highly positive: if using the tool had been optional, the majority of them would have chosen to use it without a doubt, and they would like other courses to involve a tool like the one presented in this article. A discussion about the use of this tool in subsequent years in the same and related courses is also presented, analyzing the sustainability of this new assessment method.

Keywords: assessment; assessment process; assessment tools; e-learning; assessment techniques; automated assessment; online education; computer science education

1. Introduction

The global pandemic of COVID-19 led to the suspension of face-to-face teaching activities in many countries. In the higher education context, the abrupt transformation of classroom teaching into an online format was carried out practically overnight with the aid of tools such as videoconferencing software for synchronous activities or lecture recording programs for the creation of videos that can be shared with students through learning management systems (LMSs). However, the evaluation process of some courses could not be easily transformed to an online format, since assessing the attainment of the course learning objectives is often a complex procedure that supports the whole process of teaching and learning [1]. This evaluation is the result of the previous adaptation of the courses to the European Higher Education Area (EHEA) guidelines and principles [2] aligned with the national agencies that generate the procedures, recommendations, guidelines, and support documents to implement those recommendations, such as ANECA (National Agency for Quality Assessment and Accreditation,

in its Spanish acronym) in Spain [3]. The finalist nature of the evaluation process was abandoned in favor of a new learning-oriented approach, in which feedback—which contributes to the continuous improvement of learning—gains prominence [4–6].

Carless et al. [7] established a conceptual framework for learning-oriented assessment. Carless [8] developed the concept of learning-oriented assessment itself through three characteristic elements: (1) assessment tasks should be designed to stimulate sound learning practices among students; (2) the involvement of students in the assessment process, as exemplified by the development of evaluative skills; and (3) timely feedback which feeds forward by prompting student engagement and action. This learning-oriented assessment, with its three main characteristic elements, is usually implemented in programming courses in the form of several programming assignments, along with a final face-to-face written exam in which students have to answer several theoretical and practical questions, although there is often a mid-term face-to-face exam as well [9,10]. An important distinction to highlight is that the programming assignments are used for formative assessment, and the written exams are used for summative assessment. Taras [11] analyzed both types of assessment and their characteristics, and concluded that formative assessment is in fact summative assessment combined with feedback that can be used by the learner. Although this is the main difference between summative and formative assessment, other authors—such as Harlen and James [12]—deeply characterize both types and enumerate many other differences to conclude that, although they have separate functions, they can be used together, complementing each other. Carless states that learning-oriented assessment can be achieved through either formative or summative assessment, as long as the central focus is on engineering appropriate student learning [8].

With the cancellation of classes due to COVID-19, in Spain, the Network of University Quality Agencies (REACU, in its Spanish acronym), together with ANECA, made public an agreement in which universities were requested to adopt evaluation methodologies that made the best possible use of the resources at their disposal, aligning themselves with the quality standards in force in the European Higher Education Area (EHEA), so that the following general criteria were met [13,14]:

(a) the use of different assessment methods, based on continuous assessment techniques and individual tests;
(b) these methods must enable the evaluation of the acquisition of the competences and learning outcomes of the subjects;
(c) the criteria and methods of evaluation, as well as the criteria for grading, should be made public well in advance and included in the subject teaching guides as addenda;

In the context of programming courses, meeting the aforementioned criteria required programming assignments and exams to be transformed to a fully online format. Programming assignments present several benefits for teachers and students. For instance, they can help transfer theoretical knowledge into practical programming skills and enhance student programming skills [10]. To fully realize these benefits, assessment systems for programming assignments should be used in order to grade the assignments and provide feedback to the students. These systems can be classified in the first place by assessment type. The first type is manual systems, such as the system described in [15], which assists the instructor in assessing students' assignments, but the assessment itself is performed manually by the instructor. Then, there are automated assessment systems (for example [16], which assess students' solutions automatically). Finally, semi-automated systems, such as [17], assess students' assignments automatically, but also require that the instructor perform additional manual inspections. Automated assessment systems can be also classified according to the strategy by which the assessment process is triggered. They can be student-centered, instructor-centered, or hybrid, the latter being a strategy aimed at exploring the strengths of both instructor-centered and student-centered approaches [18]. Overall, automated assessment systems have the potential to facilitate the transformation of programming assignments into an online format, especially in scenarios with a high student–teacher ratio, with the

support of other tools within a course LMS, such as the forum, notifications, or direct messages among teachers and students.

The face-to-face written exams also had to be transformed into an online format with the help of the available tools that each higher institution provided, although, in many cases, these were limited to what the LMS allowed, due to the urgency of the transformation and the lack of time to acquire new tools and train the teachers on how to use them.

Being a central part of the learning process, assessment is an important influence on students' learning and how they approach it. Entwistle [19] found that students' perception of the learning environment determines how they learn, and not necessarily the educational context in itself. According to Struyven, Dochy and Janssens [20], students approach learning differently depending on their perceptions of the evaluation and assessment, varying among a deep approach (an active conceptual analysis that generally results in a deep level of understanding), a surface approach (an intention to complete the learning task with little personal engagement, often associated with routine and unreflective memorization), and a strategic or achieving approach (an intention to achieve the highest possible grades by using well-organized and conscientious study methods and effective time management).

The recent COVID-19 pandemic was very sudden, and led to a paradigm shift in teaching and learning. Agencies have reacted quickly, creating generic recommendations and guidelines such as the ones mentioned earlier, but further research is needed to successfully and efficiently adapt specific courses to the new requirements. Addressing this research gap is especially urgent in contexts and scenarios in which the adaptation is not straightforward, such as in courses with high student–teacher ratios or in practical courses where the assessment tools or methods used in face-to-face learning are not ready to be adopted within a fully online environment.

This article presents a case study describing the transformation of the assessment method of a programming course in higher education to a fully online format during the COVID-19 pandemic by means of an automated student-centered assessment tool. To evaluate the new assessment method, we studied students' interactions with the tool, as well as students' perceptions (meaning "the way that someone thinks and feels about a company, product, service, etc." [21]), measured with two different surveys: one for programming assignments and one for the final exam. The results obtained have allowed us to analyze the sustainability of the newly developed assessment method and how it should be improved for future editions of the same course and related ones, thus filling the research gap identified earlier.

The rest of the article is organized as follows. Existing literature on automated assessment systems is reviewed in the next section. Section 3 explains the student assessment method followed in the case study presented, and its evaluation. Then, Section 4 shows and discusses the results obtained from said evaluation. Lastly, Section 5 finishes with the conclusions of the article with an outlook on future work, and Section 6 presents the limitations of the case study.

2. Related Work

Quite a number of literature reviews on automated assessment systems for programming assignments have been published over the past years [18,22–29]. These literature reviews have classified these systems according to different aspects, such as epoch [22], assessment type (automated, semi-automated, or manual) [18], analysis approach (static, dynamic, or hybrid) [27,28], assessment process triggering (student-centered, instructor-centered, or hybrid) [18], and purpose (competitions, quizzes, software testing, or non-specialized) [18]. Moreover, these literature reviews have analyzed the features offered by automated assessment systems [18,23–27]. In this regard, it is worth pointing out that, generally, these systems provide electronic submission, automated and often immediate feedback, automated grading, and statistics reporting. Pieterse [29] investigated the factors that contribute to the successful application of automated assessment systems for programming assignments, concluding that these factors include the quality and clarity of the assignments, well-chosen test data, useful feedback,

the testing maturity of students, the possibility of performing unlimited submissions, and additional support. In total, more than 100 automated assessment systems for programming assignments have been reported in the literature. Among the most popular of these systems are Mooshak [30], DOMjudge [31], CourseMarker [9], BOSS [32], WebWork [33], and Automata [34]. Automated assessment systems for programming assignments help teachers to evaluate programs written by students, and provide them with timely feedback [35]. One of the main reasons for introducing these systems in programming courses is their capacity to dramatically reduce teachers' workloads. In this regard, Bai [36] shows that these systems allow teachers to save time and, at the same time, provide quicker feedback and deliver more assignments. It must also be mentioned that these systems require a more careful pedagogical design of the student programming assignments on the part of the instructors [22,23,29,37], and that they can change how students approach these assignments [24].

Automated assessment systems are usually classified as formative assessment tools, as the feedback these tools provide usually consists of "information communicated to the learner with the intention to modify his or her thinking or behavior for the purpose of improving learning" [38], but sometimes, these tools can be considered as summative assessment tools if they only provide the grades or percentages. Most of the time, the feedback provided is a configuration option that the instructor has to provide when creating the assignments. Keuning et al. [26] conducted a systematic review of automated assessment tools with a special focus on the feedback generated.

Regarding the reliability and validity of these systems, they have been studied comparing manually graded assignments with the system generated grades. Gaudencio, Dantas and Guerrero [39] reported that instructors who manually graded assignments tended to agree more often with the grades calculated by an automated assessment system (75–97%) than with the ones provided by other instructors (62–95%). Moreover, a number of authors [9,40,41] have also reported on the grading consistency rates between automated systems and instructors, highlighting the reliability and lack of subjectivity that these systems present.

In addition to the benefits that automated assessment systems can provide for teachers, these systems can yield important benefits for students as well. Several works have evaluated the use of automated assessment systems for student programming assignments in the context of programming courses, concluding that this kind of system is capable of producing positive effects on both students' perceptions [37,42–48] and performance [47–49]. However, experiences have also been reported in which the use of these systems did not produce significant positive results. For example, Rubio-Sánchez et al. [50] evaluated Mooshak, and concluded that the generated feedback needs to be richer in order to improve student acceptance, and that there was no evidence to claim that its use helped to decrease the dropout rate. In this regard, it should be taken into account that the effectiveness of an automated assessment system in a programming course ultimately relies on how it is used and integrated into the course [29]. Therefore, it becomes clear that the teaching methodology adopted in a programming course plays a crucial role in the successful application of an automated assessment system. Evidence of this fact is that the same automated assessment system can succeed in reducing the dropout rates in a programming course [49], but fail to do so in another programming course with a different teaching methodology [50]. Although examining the effect of combining automated assessment systems for programming assignments with different teaching methodologies would be a valuable contribution, no research work has addressed this research issue yet. Moreover, no study has yet examined the use of these systems for providing student assessment methods in programming courses following the teaching methodologies adopted in response to the COVID-19 pandemic.

3. Description of the Case Study

The case study presented in this article follows the research design described by Yin [51]. Yin defines a case study as "an empirical inquiry that investigates a contemporary phenomenon (the 'case') in depth and within its real-world context". The theoretical framework of this study was introduced in the first two sections of this article. It is based on educational assessment, specifically learning-oriented

assessment (either summative or formative), and the automated assessment tools. The research gap identified was also stated as how to successfully address the adaptation of specific practical courses with high student–teacher ratios to the new requirements that the COVID-19 pandemic imposes. The real-world context in this case study is a programming course in a higher education institution in Spain that had to be urgently adapted to a fully online format due to the pandemic. The contribution of this case study is two-fold: first, it illustrates how the evaluation of a programming course in a higher education institution can be successfully transformed to the new requirements by means of a student-centered automated assessment tool, and second, it analyzes students' perceptions of the use of such a tool in the evaluation of the course and their interactions with it.

The rest of this section describes the course context as it was before the pandemic, how it was transformed due to the new requirements, and the automated student-centered assessment tool that was used for this transformation. Lastly, the instruments used to collect students' interactions with the automated assessment tool and the students' opinions on its use are detailed.

3.1. Course Context (Pre-COVID-19)

The programming course analyzed in this study is part of the Bachelor's Degree in Telecommunications Engineering at UPM (Universidad Politécnica de Madrid). It is a third-year course that accounts for 4.5 ECTS (European Credit Transfer System) credits, which is equivalent to 115–135 h of student work. In this course, the students learn the basics of web development, including HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), JavaScript, and more advanced technologies, such as node.js, express, and SQL (Structured Query Language). The course follows the AMMIL (Active Meaningful Micro Inductive Learning) Methodology [52]; hence, the complete program is recorded in video as micro-lessons.

There are nine programming assignments delivered to students through the Moodle platform used in the course. Students are required to submit all of the assignments, which account for 30% of the final grade. The remaining 70% corresponds to two written exams: one midterm and one final exam, each accounting for 35% of the final grade. In order to pass the course, students needed to achieve a grade greater than or equal to 4 out of 10 in the exams, and obtain a grade of at least 5 out of 10 in the course final grade.

There were two main reasons for introducing an automated student-centered assessment tool in this course. In the first place, the course has a high student–teacher ratio because it is a core course, and all the students pursuing the Telecommunications Engineering degree have to pass it (there are 312 students and seven teachers in total). In the second place, it is very common for students not to have high programming skills at this stage of their studies, finding it more difficult than other courses. This latter fact has three implications: in the first place, the number of programming assignments should be high (multiple and frequent small assignments instead of one final project); in the second place, students need as much detailed feedback and help as possible, which can be proven to be an additional motivating factor [53] and can improve the students' learning experience [37]. Finally, this feedback should be provided frequently and timely, even immediately (after each execution of the tool) if possible, allowing the student to continue working on the assignments with the knowledge of how well he/she is performing and learn from their mistakes, this fact has been widely studied to improve student performance and promote learning [54–56]. Summing up, a high student–teacher ratio, combined with a high number of assignments and the suitability of immediate feedback, make the use of an automated student-centered assessment tool perfect for this kind of course. Due to the fact that the feedback provided is a central piece of an automated assessment tool, three teachers were in charge of designing it after reviewing the literature, studying successful case studies and inspecting similar tools.

Although this programming course has a Moodle platform for distributing the didactic materials and relies on an automated student-centered assessment tool for the programming assignments, it also has a high face-to-face load. The teachers dedicate half of a 50 min session to explain in detail each

assignment, and one or two extra sessions to solve each of them step by step. In addition, face-to-face tutorials are frequent in this subject, in order to solve doubts and guide students in their work, as they do not have high programming skills.

3.2. Assessment Transformation

The midterm exam was scheduled for the 16th of March, and the disruption of face-to-face activities due to the COVID-19 pandemic took place on the 11th of March. Hence, following the guidelines provided by the head of studies, the midterm exam was canceled and, consequently, the final exam was worth 70% of the grade. In order to pass the course, the students needed to pass the exam (by achieving a grade greater than or equal to 5 out of 10) and obtain a grade of at least 5 out of 10 in the course's final grade, which was calculated as the weighted sum of the exam and assignments' scores.

The COVID-19 pandemic not only led to the cancelation of the midterm exam, but also to the need to transform the face-to-face lessons, the programming assignments and the final exam into a fully online format. Fortunately, as the complete program was recorded in video as micro-lessons, only some lessons and tutorials had to be conducted via videoconferencing tools. The programming assignments were planned to make use of the automated student-centered assessment tool from the beginning of the course, as explained in the previous section, but as a complementary tool with the support of the face-to-face sessions (in which the assignments were solved step-by-step) and tutorials. Now, in order to meet the new requirements, and to play a central role in the course assignments, although the automated assessment tool was already online-based, the last four programming assignments that were left when the COVID-19 situation emerged had to be adapted, explaining the problem statements in a more detailed way, and giving more elaborate feedback to students. This tool also had to be complemented with the LMS forums and videoconference sessions to substitute for face-to-face assistance. To sum things up, the automated student-centered assessment tool went from being a complementary tool in the programming assignments to being the primary tool.

On the other hand, the final exam was a major challenge, since this programming course is a very practical one, in which the skills and learning outcomes established are impossible to measure with just an online test. Additionally, students' perceptions of the assessment characteristics play a positive role in their learning, resulting in deeper learning and improved learning outcomes. Two characteristics have special influence: authenticity [57,58] and feedback [59,60]. Hence, the teaching staff of the course decided to divide the final exam into two parts: the first part—intended to measure the more theoretical concepts—was a multiple-choice test with 30 questions to be completed in 30 min, whereas the second part of the exam—intended to measure practical programming skills—was a 40-min programming assignment that made use of the same automated assessment tool used in the assignments of the course. The exam had multiple slightly different variants of a similar level of difficulty. Upon launching the assessment tool for the first time, each user was assigned a version of the exam based on their credentials. Since this procedure slightly differed from previous exercises, a mock exam was published days before the actual exam in order to help the students understand how the exam was going to be performed, and to help them practice generating the problem statement and turning it in. Both parts of the final exam accounted for 50% of the final exam grade, and each part had to be passed with at least 4 out of 10 points.

To guarantee the validity of the assessment method, all of the changes planned were included in the subject teaching guides as addenda, following the strategy made public by ANECA [14], and were notified to the students well in advance of the final exam.

3.3. Description of the Assessment Tool

This section describes the student-centered automated assessment tool used both in the programming assignments of the course and the second part of the final exam. There is a wide set of assessment tools for programming assignments available [18,22,24]. However, these tools are rarely used beyond the institutions in which they were created, because they are difficult to adapt and extend to fit new

courses [18,61]. The case study presented in this paper is another example of the latter, as the course staff had recently developed the automated student-centered assessment tool for the programming assignments, so it was easily extended and adapted to support the new requirements imposed by the COVID-19 pandemic and the interruption of the face-to-face activities.

The automated student-centered assessment tool used in this case study is called autoCOREctor, and it consists of a client and a server. The tool was designed to be easily integrated with Learning Management Systems (LMSs) and Version Control Platforms (VCPs). The LMS was used to store students' submissions and grades, which is essential for legal purposes, because learning evidence must be stored and kept in the course LMS. The VCP was employed to facilitate the management of the assignments' problem statements and check the test suites' integrity to avoid cheating. The main features of autoCOREctor are the following:

- Student-centered: students start the assessment process. They can view the assignment specification, and develop and submit a solution for it. For each submission, the tool assesses the student's solution, considering the assessment parameters provided by the instructor. After the assessment process is completed, both the instructor and the students have access to the assessment results.
- Tests are run locally on the student's computer: students can work offline, and the score they obtain locally is later uploaded to the LMS once they decide to submit it, turning it into their assignment grade.
- Unlimited number of local test runs: students can run autoCOREctor as many times as they wish, getting immediate feedback about their work, as well as their current grade.
- Unlimited number of submissions: students can submit their score and the solution of the assignment to the LMS as many times as they want in the timeframe determined by the instructor.
- Penalty for late submissions: instructors can configure a grace period in which students can submit their assignments with a penalty in their grade.
- The autoCOREctor client is distributed as an NPM (Node Package Manager) package uploaded to the official NPM repository. This feature facilitates the tool installation and update in case a new version is released by the teaching staff. The autoCOREctor client is therefore a CLI (Command Line Interface) tool that can work on any operating system—Linux, Windows or Mac—indifferently. Also, if any student has any problem with the installation, the client can be used from free online services that include a terminal, such as Glitch [62].
- Thorough documentation for instructors and students: documentation is made available to students in the LMS, and the list of available options can be listed through a shell command of the autoCOREctor client.
- Integrity check of the test suite: autoCOREctor checks that the test suite that is run to obtain the feedback and grade is the one that the instructor developed.
- Learning analytics: the autoCOREctor client logs all of the interactions that students have with it as well as the evolution of the resulting grades, uploading all this information to the autoCOREctor server for further analysis whenever students make a submission.
- Learning analytics dashboards: the autoCOREctor server generates interactive graphs of the evolution of grades per assignment and student.
- Generation of a scaffold for assignments' problem statements and a basic test suite with examples to facilitate instructors in its use in different contexts.
- Possibility to define test cases and to specify the generated feedback, as well as the way in which the grades of the assignments are calculated in multiple programming languages.
- Secure communications: all of the information that autoCOREctor sends uses secure sockets. Additionally, all of the assignments, logs, and grades are encrypted before being sent.

The following paragraphs explain the generic design of the autoCOREctor tool and then the specific deployment we have performed for this case study.

AutoCOREctor consists of a web server with a connector to send requests to the LMS, a connector to communicate with the VCP, and a client program that is executed on the students' computer. Figure 1 shows the architecture of the system from the point of view of a teacher. The details of the functionalities of each component and the interaction flow between them are as follows:

1. The teacher creates an empty repository for the assignment in the Version Control Platform.
2. The teacher creates and configures the assignment for the corresponding course in the LMS.
3. The third step is to obtain an assignment template to implement the test suite and the assignment problem statement. For this, the teacher accesses the autoCOREctor web server interface, where information about courses and assignments created by the teacher is provided. This information is retrieved by the LMS Connector which, depending on the LMS and the authentication and authorization mechanisms available, gathers it directly by the LMS API (Application Programming Interface) or using a delegated authorization mechanism such as OAuth or OpenID Connect (Step 3a). Once this is achieved, the teacher can link the repository created in Step 1 to the specific assignment of the LMS (Step 2). This link is stored in the server database, so no further interactions are needed with the LMS. As a result of this link, the server creates a template of the assignment containing the needed metadata of both the repository and the assignment, to be used later when the students submit their results. Finally, the teacher downloads this template.
4. Using the downloaded template, the teacher creates the assignment's problem statement and the test suite that will be run during its development by the students. Depending on the characteristics of each course, the tests will be made using a specific testing framework. The downloaded template contains the software libraries needed to develop such tests.
5. When the tests and the assignment problem statement are ready, the teacher uploads them to the VCP, making it available to students. If the teacher eventually fixes mistakes in the assignment or wants to create new versions, he/she only has to upload the changes to the VCP and, if the assignment is open and the students have already started working on it, notify the students to update the assignment with one Version Control System command.

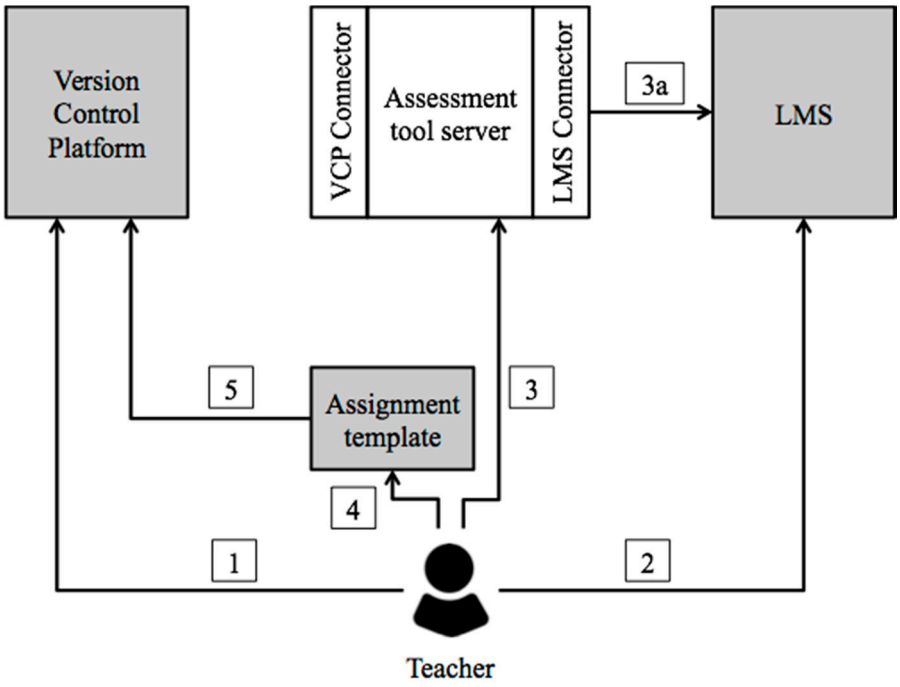

Figure 1. Automated assessment tool architecture from the point of view of the teacher.

On the other hand, Figure 2 shows the architecture from the point of view of the students. The details of the functionalities of each component and the interaction flow between them are as follows:

1. Students download an assignment from the Version Control Platform after obtaining its specific repository link from the LMS assignment. It contains the problem statement of the assignment, the template with the source code to develop it, and the test suite that the teacher has defined.
2. Before starting to develop the assignment's solution, the students have to download an authentication token from the LMS. This token will be used by the LMS Connector in Step 5 to submit the students' results to the LMS.
3. While developing the assignment, students can use the autoCOREctor client to run the tests defined by the teacher as many times as they want, receiving immediate feedback and a score each time. Every time the assessment tool client is executed, it checks whether the assignment has been updated in the Version Control Platform (Step 3a). To do this, the client has to be adapted to use the specific API of the Version Control Platform. Moreover, the client saves a history file with the evolution of the scores achieved by the student.
4. Students use this same client to submit their results to the autoCOREctor server. During this process, the server checks that the version of the tests the student is uploading is updated with the latest version at the VCP (Step 4a). Again, the VCP Connector has to be adapted to the API of the Version Control Platform. Moreover, in order to ensure the integrity and authenticity of the results, the autoCOREctor client encrypts and signs the information containing such results with a shared key with the server. Thus, when receiving the information, the server is in charge of decrypting the data and validating the signature. Furthermore, the server checks that the tests defined by the teacher have not been modified by comparing a hash code sent by the client together with the score with the same hash created from the version available in the VCP. This submission can be performed as many times as the student wants while the assignment is open, keeping the last result uploaded.
5. Finally, the server stores the file of the assignment solution, the results, and the history in its database and sends the score to the LMS using the LMS Connector. If a penalty for late submissions has been configured by the teacher in the assignment, the score is adapted accordingly in case it is necessary.

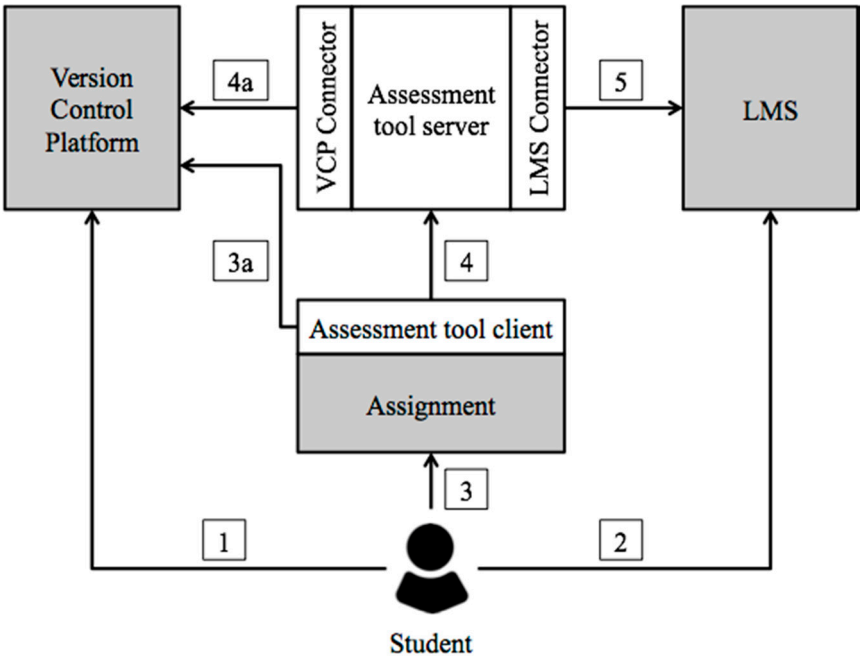

Figure 2. Automated assessment tool architecture from the point of view of the students.

Some important aspects should be commented on, relating to the reliability and validity of the assessment process. Regarding reliability, autoCOREctor performs the integrity check of the test suites

that it uses, so all of the students are assessed with the same test suite. It also signs and encrypts everything that is sent to avoid cheating. With respect to validity, before starting its use in the course, it was validated among the members of the course staff; some of them completed the assignments and some corrected the solutions, verifying that the grades were very similar. The same procedure was followed with random submissions of the final exam, verifying again that the scores reported by autoCOREctor were similar to those given by the course teachers. This verification was very informal, and since further research is needed on the validity of the grades generated by automated assessment systems, it constitutes an interesting work that will be addressed in the near future.

In our case study, we used GitHub [63] as the VCP and Moodle as the LMS. Thus, teachers create new GitHub repositories for each assignment and recieve a URL that is registered in the autoCOREctor server linked to the assignment created in Moodle. In order to retrieve the courses and assignments created by a teacher, we use the Moodle API [64], authenticating teachers by means of their username and password. As the course in which we evaluated the platform is about Node.js technology, the tool creates assignment templates with a package.json file that contains metadata about the LMS course and assignment.

In the case of this course, the tests are written by the teacher using the Mocha [65] and Zombie [66] frameworks, and the Chai library [67]. The students also need to download an authentication token from Moodle that is later used by the server to upload their scores. This token is included together with the students' email, the score, the test version, the hash for checking the integrity of the tests, and the signature in a JSON (JavaScript Object Notation) file created by the client.

3.4. Data Collection Instruments

Three instruments were used in this study in order to evaluate the students' perceptions and their interactions with autoCOREctor: (1) a survey to collect students' opinions on the use of autoCOREctor in the assignments, (2) another survey to collect students' opinions on the use of this tool in the final exam, and (3) the tool itself, which automatically records data on students' interactions with it, in order to obtain information on students' usage of the tool.

In order to collect students' perceptions on using autoCOREctor for the programming assignments in the course, a survey was conducted after the termination of the last assignment of the course. This survey was designed by the authors of this article, and was validated by three faculty members. It included some initial demographic questions, a set of closed-ended questions addressing students' general opinion and acceptance of the tool, and a list of statements with which they needed to agree or disagree using a 5-point Likert scale. These questions were aimed at assessing students' attitudes towards the use of the autoCOREctor, students' thoughts on its usability and usefulness as an assessment method, their perceptions on the feedback and grades received, their opinions on the main features of the tool, and whether they prefer it over other assessment methods. At the end of the survey, there was a space in which the students could leave suggestions, complaints, and other comments.

Moreover, with the aim of collecting students' perceptions toward using autoCOREctor for the final exam of the course, a survey was conducted after the exam. This survey was also designed by the authors of this article, and was validated by three faculty members. It included some initial demographic questions, a set of closed-ended questions addressing students' general opinions and acceptance of the activity, and a list of statements with which they needed to agree or disagree using a 5-point Likert scale. These questions were similar to the ones posed in the first survey, but they were aimed at assessing students' attitudes towards the use of the automated assessment tool in the specific context of the exam. Once again, at the end of the survey, there was a space in which the students could leave suggestions, complaints, and other comments.

Lastly, autoCOREctor automatically recorded data on student interactions with it both for the assignments and the final exam. Specifically, the following data were collected for each of the nine assignments and for the exam: the number of times each student ran the tests locally, the number of

times each student submitted their solution to the LMS, and the score they obtained in each execution of the test suite.

3.5. Data Analysis

The data gathered by the three instruments mentioned earlier were processed using Excel and Python, along with the Numpy, Scipy and Pandas software packages. A descriptive quantitative analysis was performed in the resulting datasets.

The survey data were analyzed using mean (M), standard deviation (SD), and median absolute deviation (MAD), as well as the median (MED) value, which is more representative of the central tendency in scaled non-normal distributions such as those of Likert-type variables. In addition, Cronbach's alpha (α) was calculated in order to assess the internal consistency of both surveys, confirming their reliability at $\alpha > 0.9$ for both of them. Furthermore, in view of the reduced number of responses to the open-ended questions, only an informal qualitative analysis was performed on these.

The usage data logs collected by the automated assessment tool were pre-processed in order to remove spurious logs, such as the ones generated by the faculty staff when testing the assignments. Then, the data were aggregated by assignment, and by whether the log corresponded to a local use of the tool or a submission. The resulting dataset includes the number of average executions (local and submissions) per student and assignment, as well as the total executions per assignment. This information allowed us to analyze the evolution of the usage pattern of autoCOREctor throughout the course, as well as the workload to which autoCOREctor is subjected.

4. Results and Discussion

4.1. Results of the Student Survey on the Use of the Automated Assessment Tool on Programming Assignments

Table 1 shows the results of the student survey conducted after carrying out the programming assignments of the course, including, for each question, the mean (M), median (MED), standard deviation (SD), and median absolute deviation (MAD), along with the number of answers (N). The survey was completed by 85 students (65 men and 20 women), with a median age of 21 (MAD = 1.0). Figure 3 shows the distribution of the students' responses.

The results of the survey show that the students had a positive overall opinion of the use of the autoCOREctor automated assessment tool in the programming assignments of the course (MED = 4.0, MAD = 1.0). In terms of usability, the students strongly believe autoCOREctor was very easy to install (MED = 5.0, MAD = 0.0) and to use (MED = 5.0, MAD = 0.0). These aspects are of special relevance, since many students find programming difficult to learn, as evidenced by the high failure rates that programming courses usually have [68]; thus, using the assessment tool should not pose yet another difficulty for them.

Students' opinions diverged when asked whether they agreed with the survey statements that the feedback provided by autoCOREctor was useful (MED = 3.0, MAD = 1.0), that it was easy to understand (MED = 4.0, MAD = 1.0), and that it helped them to improve their assignments (MED = 3.0, MAD = 1.0). One possible reason why not all of the students found the feedback provided by autoCOREctor useful is probably that it merely pointed out the errors found in their solutions, but did not tell them how to fix them, which students would have found to be of more utility, despite its being detrimental to their learning. However, the most likely reason why they did not find the feedback provided by the tool extremely useful is that the students often make syntax mistakes in their code, which can prevent autoCOREctor from functioning as expected and cause it to throw default error messages that can be somewhat cryptic, instead of displaying the ones provided by the teachers for each specific aspect of the assignment that is not working properly. In this regard, previous studies have also found that students have difficulties understanding the feedback provided by automated assessment systems [45], and some of them concluded that the quality of the feedback needs to be improved in order to increase student acceptance [44,50]. Nonetheless, in this study, most students were very

confident that they would rather receive the feedback provided by autoCOREctor than no feedback at all (MED = 5.0, MAD = 0.0), and that if using the tool had been optional, the majority of them would have chosen to use it without a doubt (MED = 5.0, MAD = 0.0), providing more evidence that, overall, autoCOREctor was useful for students. In fact, most of them agreed that autoCOREctor helped them discover errors in their assignment that they did not know they had (MED = 4.0, MAD = 1.0), which, without autoCOREctor, would only have been possible through manual teacher assessment and, consequently, infeasible in such a crowded course.

Table 1. Results of the student survey on the use of autoCOREctor on programming assignments.

Question	N	M	MED	SD	MAD
Q1. What is your general opinion on autoCOREctor on a scale of 1 (Very bad) to 5 (Very good)?	85	3.9	4	0.9	1.0
State Your Level of Agreement with the Following Statements on autoCOREctor on a Scale of 1 (Strongly Disagree) to 5 (Strongly Agree)					
Q2. autoCOREctor was easy to install	85	4.5	5.0	0.9	0.0
Q3. autoCOREctor was easy to use	85	4.3	5.0	1.0	0.0
Q4. The feedback provided by autoCOREctor was useful	85	3.3	3.0	1.1	1.0
Q5. The feedback provided by autoCOREctor was easy to understand	85	3.3	4.0	1.1	1.0
Q6. The feedback provided by autoCOREctor helped me improve my assignments	85	3.5	3.0	1.1	1.0
Q7. I'd rather receive the feedback provided by autoCOREctor than no feedback whatsoever	85	4.6	5.0	0.9	0.0
Q8. If using autoCOREctor had been optional, I would have chosen to use it without a doubt	85	4.6	5.0	0.9	0.0
Q9. autoCOREctor has helped me discover errors in my assignments that I did not know I had	85	3.8	4.0	1.1	1.0
Q10. Having to pass the autoCOREctor tests for each assignment has made me spend more time in the assignments than I would have if I had not used it	79	2.9	3.0	1.4	1.0
Q11. autoCOREctor has increased my motivation to work on the assignments	81	3.5	4.0	1.0	1.0
Q12. Being able to run the autoCOREctor tests repeated times with no penalty and obtaining instant feedback has made me invest more time in the course assignments	83	4.3	5.0	1.0	0.0
Q13 The grades provided by autoCOREctor were fair	85	4.3	5.0	1.0	0.0
Q14. Thanks to autoCOREctor I think I got a better grade in the assignments that I would have without it	85	4.6	5.0	1.0	0.0
Q15. In general, using autoCOREctor has improved my programming knowledge	84	3.9	4.0	1.1	1.0
Q16. I think by using autoCOREctor I have improved my programming knowledge more than I would have with a manual assessment	84	4.0	4.0	1.1	1.0
Q17. In general, I believe the use of automated assessment tools such as autoCOREctor improves the evaluation process of the course assignments when compared to the classical manual procedure	85	4.6	5.0	0.9	0.0
Q18. I would like to have tools like autoCOREctor in other courses	84	4.7	5.0	0.8	0.0
Indicate How Useful you Find Each of the Following Features of autoCOREctor on a Scale of 1 (Useless) to 5 (Very Useful)					
Q19. It allows to run the tests an unlimited number of times	84	4.9	5.0	0.5	0.0
Q20. It has a command to directly upload the assignment to Moodle	84	4.7	5.0	0.8	0.0
Q21. It allows to run the tests locally on the student's computer	83	4.6	5.0	0.8	0.0
Q22. It provides instant feedback each time the tests are run	84	4.4	5.0	1.0	0.0
Q23. It provides documentation to learn how to use it and the options it provides	77	3.9	4.0	1.0	1.0

On another note, the students' opinions diverged regarding whether having to pass the tests suites provided by autoCOREctor for each assignment made them spend more time working on them than they would have without the tool (MED = 3.0, MAD = 1.0). However, they strongly agreed with the fact that being able to run the tests repeatedly and obtaining instant feedback made them invest

more time (MED = 5.0, MAD = 0.0). On the one hand, since students know their grade at every step of the way while working on an assignment, they can see how more work translates into a higher grade in real-time, encouraging them to keep working and dedicating time to the assignment, and to strive for the best score. In this regard, the students somewhat agreed that autoCOREctor has increased their motivation to work on the assignments (MED = 4.0, MAD = 1.0). On the other hand, once they reach the maximum score, they stop working on the assignments and turn them in, whereas, without the automated assessment tool, they keep testing each assignment manually until they are sure it works properly, which takes considerably more time. Hence, since the feedback provided by the tool helps them to identify mistakes, they do not spend as much time testing. Despite the fact that this is generally regarded as a negative aspect of automated assessment tools, the programming course analyzed in this study is focused on implementation and does not cover software testing, so the students do not have the knowledge required to properly develop testing suites for their code anyway. Thus, besides manually testing their solutions, autoCOREctor is the only resource they have to check if their code is correct. In addition, the feedback provided by autoCOREctor does not reveal how to fix mistakes, but instead merely points them out, so they still need to figure out how to fix them by themselves, which also takes considerable time. It should be noted that, even though the students do not know how to perform software tests, they are taught how to debug their code using standard development tools so that they can track down the errors identified by autoCOREctor. In sum, although is not clear if using autoCOREctor requires students to invest more or less time in the assignments, the results suggest that they do not spend as much time manually looking for errors as they would without the tool but, thanks to the instantaneous feedback, they spend more time working on the solution of the assignment itself and debugging the errors pointed out by the tool.

Moreover, most of the students strongly believed that the grades provided by autoCOREctor were fair (MED = 5.0, MAD = 0.0). They also very strongly agreed with the statement that—thanks to the automated assessment tool—they got a better grade in the assignments than they would have without it (MED = 5.0, MAD = 0.0), which was an expected outcome, since the instant feedback and unlimited number of attempts allowed them to progressively improve their grades until they were content. On the one hand, since—by using autoCOREctor—they spent more time working on the assignments and obtained a better grade than they would have without the tool, it is not surprising that they found the grades provided by the tool fair. On the other hand, it could be argued that autoCOREctor is a little too sensitive to typing errors and mishaps since, with just a slight syntax error in the code, the tool would provide a grade of 0, even if all of the features were correctly implemented. However, this limitation did not impact the students' opinions in this case study. Hence, it is likely that the students' positive perceptions on the grades provided by autoCOREctor were a result of the unlimited number of attempts; thus, their opinions apply not to the score calculated by autoCOREctor each time they ran the tests, but rather to the grade they ultimately obtained for each assignment in the course, thanks to the unlimited number of attempts and the instant feedback that allowed them to fix the errors and achieve the best possible score. Consequently, the authors believe that, if the number of attempts has been limited, students' perceptions on grading fairness would have been much more negative.

With regard to self-efficacy (i.e., students' perceived skills [69]), students reported that they have improved their programming knowledge by using autoCOREctor (MED = 4.0, MAD = 1.0), even more than they would have with manual assessment (MED = 4.0, MAD = 1.0). These were the expected outcomes, since autoCOREctor allowed them to recieve feedback on their actions right away, helping them learn from their mistakes along the way. In this regard, it is worth mentioning that the piece of feedback that the students receive from autoCOREctor is not only a number, which is usually the case in manual assessment in large-enrollment courses, but is rather a message that tells them what is wrong or missing for each part of the assignment, inviting them to fix it. Moreover, the students strongly believe that the use of automated assessment tools, such as autoCOREctor, improves the evaluation process of the course assignments when compared to a manual procedure consisting in manually submitting the code to the LMS and waiting for the teacher to correct it after the deadline

(MED = 5.0, MAD = 0.0), which is the usual procedure in most courses. Thus, the fact that the feedback provided by autoCOREctor is instantaneous and actually feed-forward (i.e., feedback that lets students act upon it, giving them a chance to improve their solutions) is regarded highly by the students, since it allowed them to learn from their mistakes, as opposed to traditional feedback, which is commonly not provided in a timely manner, as students usually receive it when they have already forgotten about how they reached their solution and they are often not allowed to improve their work on the basis of the feedback received. In essence, it is safe to conclude that the students are satisfied with the automated assessment tool in this course, and that they would very much like to have similar tools in other courses (MED = 5.0, MAD = 0.0).

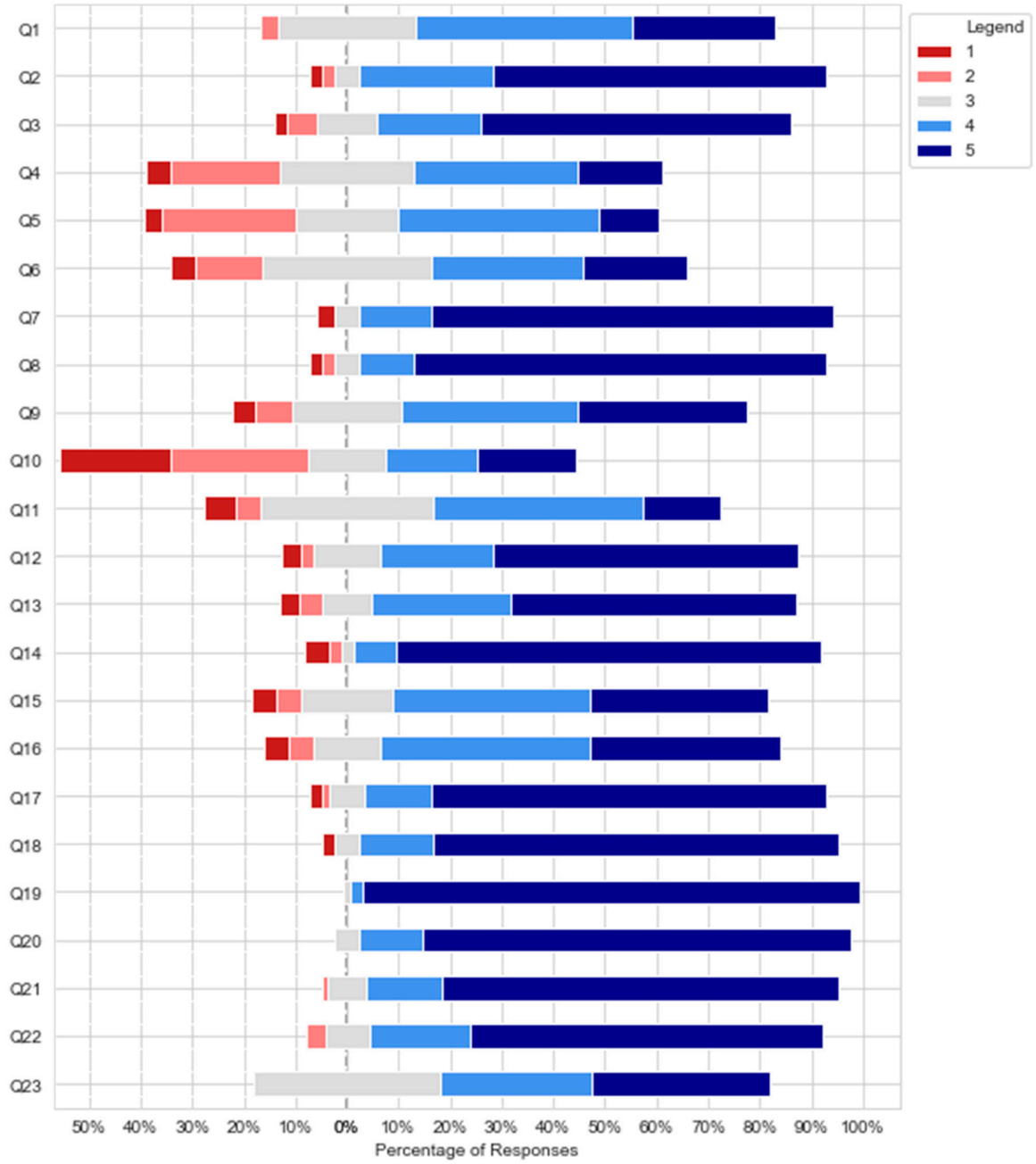

Figure 3. Distribution of responses to the student survey on the use of autoCOREctor on programming assignments.

When asked to rate the different features of autoCOREctor, the students showed a very strong preference for the ability to run the tests an unlimited number of times (MED = 5.0, MAD = 0.0), which came as no surprise since, each time the tests were executed, the tool provided them with their score and a piece of instantaneous custom feedback, as mentioned earlier. The second most highly rated aspect was the command to directly upload the assignment to the LMS (MED = 5.0, MAD = 0.0). Since this command allowed the students to submit their assignments automatically, without directly interacting with the LMS, it was very convenient for them, and prevented them from making mistakes when packaging and uploading the assignment. The students highly valued being able to run the tests locally on their computer (MED = 5.0, MAD = 0.0), since not having to upload the assignment to a remote server each time they want to receive feedback saves a lot of time and avoids errors derived from running the code in different execution environments. Predictably, the students highly appreciated the instant feedback received each time the tests were run, as discussed earlier (MED = 5.0, MAD = 0.0). Lastly, the students also found that the documentation provided to help them install and use autoCOREctor was useful (MED = 4.0, MAD = 1.0). Since the tool is so easy to install and use, the documentation was not needed in most cases, so it comes as no surprise that some students might not have used it, or found it superfluous. As can be seen in Figure 3, overall, the students' opinions on all of the features of autoCOREctor are very positive, and there are barely any negative responses for any of them.

In the last field of the survey—reserved for complaints, suggestions and other comments—the students confirmed that autoCOREctor motivated them to keep trying to get a better score and made working on the assignment more fun. Moreover, they also stated that autoCOREctor was useful when finding the mistakes they made along the way. Some students complained about the quality of the feedback received, saying it was too generic. It should be noted that this feedback depends on the specific test suite that is being executed, and is not a limitation of the automated assessment tool itself. Moreover, giving feedback that is too specific, telling students how to fix their mistakes, would prevent them from reaching the solution by themselves. As mentioned, although the feedback provided by autoCOREctor informs students of what is wrong or missing in their assignments, they still need to find which part of their code is erroneous or incomplete, and figure out how to fix it. This is indeed a crucial part of learning programming since it promotes self-assessment, making students develop the highest level of Bloom's taxonomy, which deals with evaluation [70].

4.2. Results of the Student Survey on the Use of the Automated Assessment Tool in the Final Exam

The results of the student survey conducted after using autoCOREctor in the final exam of the course are shown in Table 2, including, for each question, the mean (M), median (MED), standard deviation (SD), and median absolute deviation (MAD), along with the number of answers (N). The survey was completed by 45 students (35 men and 10 women), with a median age of 21 (MAD = 1.0). Figure 4 shows the distribution of the students' responses.

According to the students, autoCOREctor was easy to use in the exam (MED = 4.0, MAD = 1.0). In fact, one of the main reasons that encouraged the faculty staff to use autoCOREctor in the exam was precisely that students were already accustomed to this tool, and thus incorporating it into the exam would not bring any additional difficulty to those students who spent time using autoCOREctor throughout the course. The main difference between the use of autoCOREctor in the assignments and in the exam was that, in the exam, the tool provided a different problem statement for each student from among several options, making it more difficult for them to copy from one another. In this regard, the students stated that generating their problem statement was very easy as well (MED = 5.0, MAD = 0.0).

Regarding feedback, the students' opinions were similar to the ones in the previous survey. It should be mentioned that autoCOREctor was not used in the final exam merely as an instrument to measure students' competence, but rather it allowed the transformation of the exam (traditionally a summative assessment activity) into a learning-oriented assessment activity, since the students were

provided with the same type of feedback as in the programming assignments as well as an unlimited number of attempts to solve the exam. The only additional constraints of using autoCOREctor in the exam compared to its use in the assignments, apart from the different problem statements mentioned earlier, were that the students were given 40 min to complete their task (instead of several weeks), and that they could not speak to each other during the exam. When asked about the feedback provided by autoCOREctor during the exam, most of the students perceived it as useful (MED = 4.0, MAD = 1.0) and easy to understand (MED = 4.0, MAD = 1.0), and stated that it helped them to improve their solution (MED = 4.0, MAD = 1.0), although there were many students that thought otherwise. One possible explanation is that, since the exam was similar to the programming assignments, the students that had reviewed for the exam were confident of their solution, and used autoCOREctor only to verify that they were not forgetting to implement any feature and that they did not have any typos that could cause the autoCOREctor tests to fail. In fact, once again, using autoCOREctor allowed some of them to discover errors in their exam that they did not know they had (MED = 4.0, MAD = 1.0). By finding out the errors and missing features in their solutions before submitting, the students were able to use the knowledge they already had to amend those issues. This chance to incrementally improve their solutions is usually not given to students in final exams, at least definitely not in paper-based ones. Thus, using autoCOREctor allowed the students not to lose points because of a missing feature or small mistake, and thus they obtained a grade that was more representative of their actual knowledge, and that reflects their acquired skills better than a classic paper-based exam would. In fact, as far as grades are concerned, the students thought the grades provided by autoCOREctor were fair (MED = 5.0, MAD = 0.0), and they believed that they obtained a somewhat better grade in the practical final exam than if they had not used the tool (MED = 4.0, MAD = 1.0), since they would not have had any sort of feedback during the exam, and would have had to rely solely on their manual tests. That is probably one of the reasons why they stated that, if using autoCOREctor had been optional in this test, most of them would have chosen to use it anyway (MED = 5.0, MAD = 0.0).

Table 2. Results of the student survey on the use of the automated assessment tool in the final exam.

Question	N	M	MED	SD	MAD
State Your Level of Agreement with the Following Statements on autoCOREctor on a Scale of 1 (Strongly Disagree) to 5 (Strongly Agree)					
Q1. autoCOREctor was easy to use	45	4.0	4.0	0.9	1.0
Q2. Generating my problem statement was easy	45	4.6	5.0	0.7	0.0
Q3. The feedback provided by autoCOREctor was useful	43	3.8	4.0	1.1	1.0
Q4. The feedback provided by autoCOREctor was easy to understand	43	3.6	4.0	1.1	1.0
Q5. The feedback provided by autoCOREctor helped me improve my solution to the exam	42	3.5	4.0	1.4	1.0
Q6. autoCOREctor has helped me discover errors in my exam that I did not know I had	43	3.5	4.0	1.4	1.0
Q7. The grades provided by autoCOREctor were fair	45	4.2	5.0	1.3	0.0
Q8. Thanks to autoCOREctor I got a better grade in the practical final exam than if I had not used autoCOREctor	45	3.8	4.0	1.4	1.0
Q9. If using autoCOREctor had been optional in this exam, I would have chosen to use it	43	4.4	5.0	1.0	0.0
Q10. I think autoCOREctor is an adequate tool for a practical programming exam	45	4.1	4.0	1.0	1.0
Q11. The final exam using autoCOREctor adequately assesses the competences attained during the course	45	3.8	4.0	1.1	1.0
Q12. autoCOREctor assesses practical competences better than quiz-based tests or open-ended questions	44	4.2	4.0	0.9	1.0
Q13. I prefer to take a practical test using autoCOREctor than an oral exam via videoconference	45	4.6	5.0	0.8	0.0
Q14. If there were a face-to-face practical final exam in the computer laboratory, I would like it to be based in autoCOREctor	42	4.5	5.0	0.8	0.0
Q15. I would like the practical exams of other courses to involve a tool like autoCOREctor	45	4.2	4.0	1.0	1.0

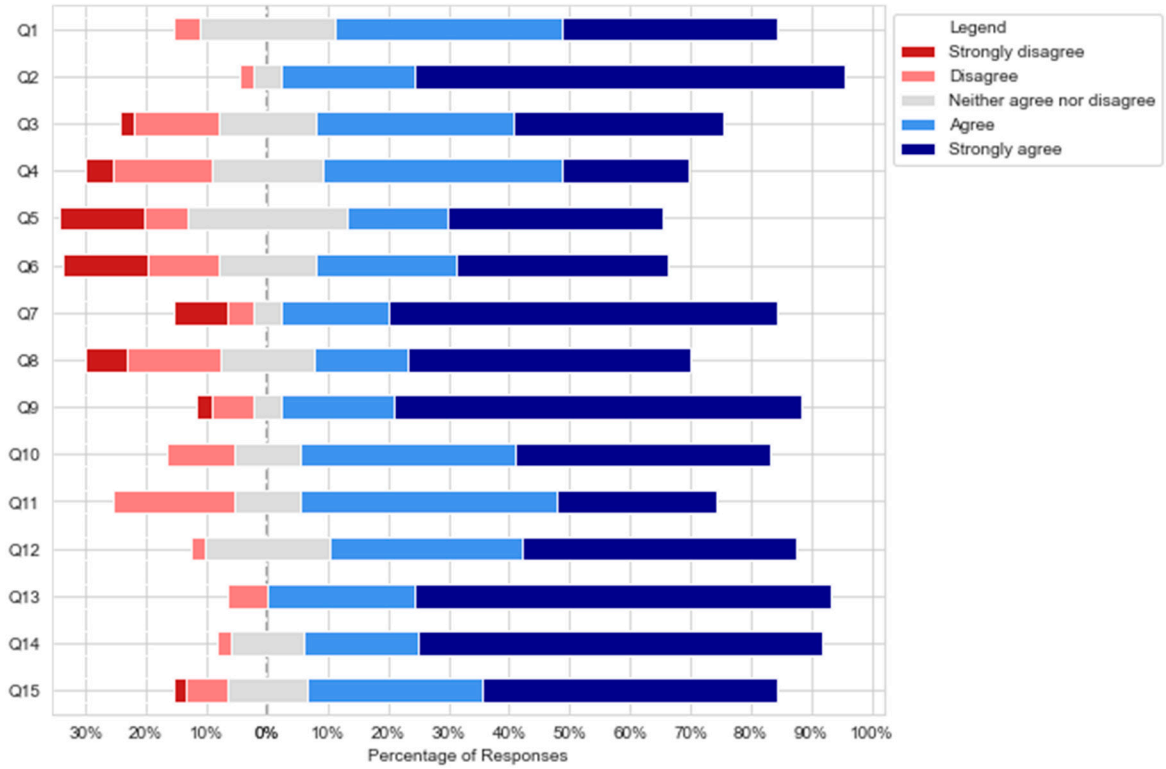

Figure 4. Distribution of responses to the student survey on the use of autoCOREctor in the final exam.

Overall, the results of the survey show that the students think autoCOREctor is an adequate tool for a practical programming test (MED = 4.0, MAD = 1.0). The students somewhat agree with the statement that the final exam powered by this tool adequately assessed the competences attained during the course (MED = 4.0, MAD = 1.0), at least more so than quiz-based tests or open-ended questions (MED = 4.0, MAD = 1.0), which in itself is a great outcome, since it was the main aim of using autoCOREctor. Most of the students also strongly agreed that they prefer taking a practical test using autoCOREctor, rather than an oral exam via videoconference (MED = 5.0, MAD = 0.0), probably because they do not undergo so much pressure as in the latter. Furthermore, the majority of students strongly agreed that, in the event of a face-to-face practical final exam in the computer laboratory, they would like it to be based on autoCOREctor as well (MED = 5.0, MAD = 0.0). These results highlight that the assessment method used in the course is not only adequate for distance scenarios, but for face-to-face ones as well. This is of special relevance, since there is very little certainty as to what the future has in store regarding going back to school or keeping the remote model, so adopting flexible solutions is the key to being able to switch from one scenario to another overnight. Moreover, this approach is easily transferable to other disciplines. In fact, most of the students agreed that the practical exams of other courses should involve a tool like autoCOREctor (MED = 4.0, MAD = 1.0).

In the space reserved for comments, suggestions, and complaints, the students reiterated that they were satisfied with autoCOREctor. However, some of them brought up the issues regarding feedback that were discussed in the previous subsection. In fact, one student said that the limited time available in the exam prevented him from finding the errors pointed out by autoCOREctor. Thus, if feedback was important for students in programming assignments, in the context of a final exam, in which time is limited, the importance of feedback is even greater, since it can make a huge difference in students' grades, and in their overall experience of using the tool during the exam. The amount of feedback provided should be carefully selected in order to let students know that something is wrong or missing, whilst also allowing them to realize their own mistakes and fix them.

4.3. Usage Data Collected by the Automated Assessment Tool

In order to study students' usage pattern of autoCOREctor, the data collected by the tool itself are provided in Table 3. These data include, for each of the nine programming assignments, for the exam, and overall: the average number of local executions of autoCOREctor per student, the total number of local executions of autoCOREctor overall, the average number of submissions per student, and the total number of submissions overall.

Table 3. Students' usage data of the automated assessment tool.

Assignment	Average Local Executions per Student	Total Local Tests	Average Submissions per Student	Total Submissions
Assignment 1	22.6	6605	1.1	331
Assignment 2	10.2	3056	1.1	316
Assignment 3	16.5	4623	1.2	329
Assignment 4	10.9	3059	1.1	303
Assignment 5	21.1	5982	1.3	361
Assignment 6	11.0	2997	1.1	295
Assignment 7	9.1	2711	1.2	334
Assignment 8	7.3	2954	1.1	306
Assignment 9	8.2	2092	1.1	285
Overall Assignments	13.1	3787	1.1	318
Final Exam	14.2	2947	2.2	441

As can be extracted from the data, the number of local executions of the test suites varied greatly among the different assignments, since each one of them had a different level of difficulty and number of features that students needed to develop. The first assignment was the one with more test executions, which is an expected result, since the students were still getting acquainted with autoCOREctor. When the COVID-19 pandemic struck, the students were working on Assignment 5. The data show that the number of executions on this particular assignment increased compared to the preceding and subsequent ones, but since then the trend was downwards. In view of the great differences among the number of executions of the different assignments, this decline cannot be attributed to the pandemic, since it might be due to other causes, such as the growing programming expertise of the students, for instance. Overall, the number of local executions of the test suites adds up to 3787 per assignment on average (13.1 per student on average), and 2947 (14.2 per student on average) in the final exam. These figures represent the number of times that the students' assignments were evaluated by autoCOREctor, which is by no means close to what it would be with manual assessment. In addition, the fact that the test suites run in the students' computers makes the tool more scalable, since all of the computational load needed to run the tests is removed from the server, which only needs to handle submissions.

On another note, the average number of submissions per student and assignment is close to one. This is the number of times that the students turned in their assignments once they were satisfied with their grades. As can be seen, in the programming assignments, the students mostly waited until they got their desired grade before submitting, whereas in the case of the final exam, this number doubled. The reason for this is that, during the exam, the students made a submission as soon as they got a passing grade in order to secure it and, after that, they kept working towards achieving a better score, and submitted their work again before the time was up. Overall, the data reveal that the students made extensive use of autoCOREctor, and that the design of the tool made it possible to withstand the high demand of the students.

5. Conclusions and Future Work

This article presents a case study describing the transformation of the assessment method of a programming course in higher education into a fully online format during the COVID-19 pandemic,

by means of a student-centered automated assessment tool called autoCOREctor. This tool was recently developed by the teaching staff of the programming course used in this case study, so it was easily extended and adapted to be used for both the programming assignments and the final exam under the new requirements that the COVID-19 pandemic imposed. The use of a student-centered automated assessment tool, not only for the assignments but also for the final exam, constitutes a novel contribution of this article, and could help the teachers of related courses to take the same approach under the same or similar circumstances. To evaluate the new assessment method, we studied students' interactions with the tool, as well as students' perceptions, as measured with two different surveys: one for the programming assignments and one for the final exam. The results show that students' perceptions of the assessment tool were positive. Previous research works [57–60] state that students' perceptions play a positive role in their learning, resulting in deeper learning and improved learning outcomes. Two assessment characteristics have a special influence on students' effective learning: authenticity [57,58] and feedback [59,60]. Regarding the former, the students stated that autoCOREctor assessed practical competences better than quiz-based tests or open-ended questions, and, in general, that using autoCOREctor improved their programming knowledge. When asked about the latter, they stated that the feedback provided was useful and easy to understand, and that they would rather receive the feedback provided by autoCOREctor than no feedback whatsoever. One important concern should be stated about authenticity: using an automated assessment tool is not as authentic as a practical exam where students do not have any feedback or help from an assessment tool and they have to debug their programs to find their errors. This is something that should be considered by the teaching staff before using this kind of tool. In this case study, it was an easy decision, as in this context the students do not have high programming skills, and software testing is out of the scope of the course.

Based on the evidence that is presented in this study, it can be suggested that student-centered automated assessment systems can be a great help for students when appropriately integrated into the teaching method of the course. The students stated that, if using the tool had been optional, they would have chosen to use it without a doubt, and they would like other courses to involve a tool like autoCOREctor. Furthermore, they assert that they dedicated more time to the assignments and that they obtained better grades thanks to the tool. Finally, the generated grades were considered fair by them, both in the exam and in the programming assignments. On the one hand, this is an important result, as fairness has proven to be positively correlated with student motivation and effective learning [71], but, on the other hand, it should be further researched, as in our case study we have not delved deeper into it.

Since autoCOREctor is a versatile tool that can be adapted to different scenarios, in this case study, we took advantage of this versatility to be able to urgently change the course assessment method and adapt it to a fully online format in a timely manner. Programming assignments and exams are part of the course assessment, but they have different characteristics; the former is a formative assessment and the latter is a summative assessment, the main difference between them being whether the student receives feedback and how elaborate this feedback is [11]. By using autoCOREctor, the teacher is the one in charge of writing the feedback received by students, which can be very detailed, revealing to the student how to solve the problem found or where to look for a possible solution (which is more adequate in the programming assignments), or it can be more scarce, only showing the error found and letting students apply their knowledge to solve the problem (which is more suited to exams).

Regarding the use of an automated student-centered assessment tool in the exam, the experience reported in this article constitutes another original contribution to the existing body of knowledge, as no work has been found in the literature reporting this specific use of an automated assessment system. For this particular use, several advantages and disadvantages were identified in this case study. In the first place, the main advantage is that it enables the assessment of practical competences for crowded courses in remote scenarios that could not be measured with an online test alone, applying the same assessment criteria to all of the students taking the exam. The tool allows the teachers of the

course to control and monitor the whole assessment process, providing them with learning analytics that can be used to improve future editions, which is a great advantage. This advantage can also pose a disadvantage, since the whole exam relies on the tool and, if it fails, the exam cannot be completed. To mitigate this drawback, a mock exam should be carried out days before the real exam, and an alternative submission method should be enabled in the Moodle platform in case the tool server hangs up or freezes. Additionally, a videoconference room can be facilitated for students during the exam, in order to solve any technical problem they might find. Another great advantage is that it has proven to be flexible enough to be used in both scenarios—face-to-face and fully online—making it a very adequate tool to be used over the next few years, in which the COVID-19 pandemic is still threatening to interrupt face-to-face activities again. In Spain, the Ministry of Universities has distributed some recommendations and guidelines to adapt next year's course to what they call the 'new normal' in the presence of COVID-19 [72]. These recommendations include an enhanced digitization strategy and teaching method adaptability, being able to switch from face-to-face to remote overnight. One final characteristic identified which could bring an advantage is that, since the autoCOREctor client is executed on students' computers, the grade is generated in said environment, avoiding software differences between the student's environment and the assessment environment, which could otherwise make the grade different between both scenarios (as might happen in instructor-centered assessment tools). However, at the same time, this can pose a security issue if students hack the tool and obtain a grade without solving the problem statements that were set out. The teaching staff has not identified any security pitfalls to date, but this could be an interesting future work to analyze it through.

With all these concerns in mind, the teaching staff of the course is convinced of using it over the coming years, regardless of whether classes are face-to-face or remote. Moreover, if the pandemic does not subside, and its requirements are sustained over time, the use of automated assessment systems for both assignments and exams can play an important role in practical courses.

Since autoCOREctor has proven to be an effective tool for use in a crowded programming course at a higher education institution, another interesting future research topic would be to analyze its involvement in the assessment of the most crowded courses nowadays: MOOCs (Massive Open Online Courses). Special attention should be paid to students' perceptions of the tool and their use of it. Besides this, in MOOCs about programming, the use of student-centered automated assessment tools as an alternative to the traditional method of assessment in general-purpose MOOCs (i.e., online tests and peer-to-peer evaluation) could be further researched, determining whether the use of these tools can bring any advantage and provide an effective assessment method.

6. Limitations

Several limitations of this case study should be noted. First of all, the surveys were only validated for internal consistency using Cronbach's Alpha, and were reviewed by the members of the course staff (several of them e-learning experts), but not further validations—such as principal components analysis or a pilot test—were performed. Second of all, as happens with all home assignments, although the student introduces his/her private credentials (i.e., email and token) in the tool, it cannot be ensured that the student is the one doing the assignment. The methods to assure this might also constitute an interesting future work. Finally, additional and more robust conclusions could be drawn if the tool were used in another scenario or context, such as another practical course that allowed us to obtain comparable data.

Author Contributions: Conceptualization: E.B., S.L.-P., Á.A., A.G.; software: E.B., S.L.-P., Á.A.; validation: E.B., S.L.-P., Á.A., J.F.S.-R., J.Q.; data curation: S.L.-P.; writing—original draft preparation: E.B., S.L.-P., Á.A., A.G.; writing—review and editing: E.B., S.L.-P., Á.A., J.F.S.-R., A.G.; supervision: J.Q. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. García-Peñalvo, F.J.; Corell, A.; Abella-García, V.; Grande, M. Online Assessment in Higher Education in the Time of COVID-19. *Educ. Knowl. Soc.* **2020**. [CrossRef]
2. Zhang, L.-Y.; Liu, S.; Yuan, X.; Li, L. Standards and Guidelines for Quality Assurance in the European Higher Education Area: Development and Inspiration. In Proceedings of the International Conference on Education Science and Development (ICESD 2019), Shenzhen, China, 19–20 June 2019.
3. ANECA. *Guía de Apoyo para la Redacción, Puesta en Práctica y Evaluación de los Resultados del Aprendizaje*; ANECA: Madrid, Spain, 2020. Available online: http://www.aneca.es/content/download/12765/158329/file/learningoutcomes_v02.pdf (accessed on 24 August 2020).
4. McAlpine, M. *Principles of Assessment*; CAA Centre, University of Luton: Luton, UK, 2002; ISBN 1-904020-01-1.
5. Nicol, D.; MacFarlane-Dick, D. Formative assessment and selfregulated learning: A model and seven principles of good feedback practice. *Stud. High. Educ.* **2006**, *31*, 199–218. [CrossRef]
6. Luo, T.; Murray, A.; Crompton, H. Designing Authentic Learning Activities to Train Pre-Service Teachers About Teaching Online. *Int. Rev. Res. Open Distrib. Learn.* **2017**, *18*, 141–157. [CrossRef]
7. Carless, D.; Joughin, G.; Liu, N. *How Assessment Supports Learning: Learning-Oriented Assessment in Action*; Hong Kong University Press: Hong Kong, China, 2006.
8. Carless, D. Learning-oriented assessment: Conceptual bases and practical implications. *Innov. Educ. Teach. Int.* **2007**, *44*, 57–66. [CrossRef]
9. Higgins, C.A.; Gray, G.; Symeonidis, P.; Tsintsifas, A. Automated Assessment and Experiences of Teaching Programming. *ACM J. Educ.l Resour. Comput.* **2005**, *5*, 5. [CrossRef]
10. Robins, A.; Rountree, J.; Rountree, N. Learning and teaching programming: A review and discussion. *Int. J. Phytoremediat.* **2003**, *21*, 137–172. [CrossRef]
11. Taras, M. Assessment—Summative and formative—Some theoretical reflections. *Br. J. Educ. Stud.* **2005**, *53*, 466–478. [CrossRef]
12. Harlen, W.; James, M. Assessment and learning: Differences and relationships between formative and summative assessment. *Int. J. Phytoremediat.* **1997**, *21*, 365–379. [CrossRef]
13. REACU. *Acuerdo de REACU de 3 de abril de 2020, ante la Situación de Excepción Provocada por el COVID-19*; REACU: Madrid, Spain, 2020.
14. ANECA. *Estrategia de ANECA para el Aseguramiento de la Calidad en la Enseñanza Virtual*; ANECA: Madrid, Spain, 2020.
15. Dawson-Howe, K.M. Automatic Submission and Administration of Programming Assignments. *ACM SIGCSE Bull.* **1995**, *27*, 51–53. [CrossRef]
16. Jackson, D. A semi-automated approach to online assessment. In Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education—ITiCSE '00, Helsinki, Finland, 10–14 July 2000; ACM Press: New York, NY, USA, 2000; pp. 164–167.
17. Blumenstein, M.; Green, S.; Nguyen, A.; Muthukkumarasamy, V. GAME: A generic automated marking environment for programming assessment. In Proceedings of the International Conference on Information Technology: Coding Computing, ITCC, Las Vegas, NV, USA, 5–7 April 2004; pp. 212–216.
18. 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, CSEEandT 2016, Dallas, TX, USA, 6–8 April 2016; pp. 147–156.
19. Entwistle, N.J. Approaches to learning and perceptions of the learning environment—Introduction to the Special Issue. *High. Educ.* **1991**, *22*, 201–204. [CrossRef]
20. Struyven, K.; Dochy, F.; Janssens, S. Students' perceptions about evaluation and assessment in higher education: A review. *Assess. Eval. High. Educ.* **2005**, *30*, 325–341. [CrossRef]
21. Perception Definition in the Cambridge Dictionary. Available online: https://dictionary.cambridge.org/dictionary/english/perception (accessed on 24 August 2020).
22. Douce, C.; Livingstone, D.; Orwell, J. Automatic Test-Based Assessment of Programming: A Review. *ACM J. Educ. Resour. Comput.* **2005**, *5*, 4. [CrossRef]
23. Ala-Mutka, K.M. A survey of automated assessment approaches for programming assignments. *Comput. Sci. Educ.* **2005**, *15*, 83–102. [CrossRef]

24. 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; ACM Press: New York, NY, USA, 2010; pp. 86–93.
25. Caiza, J.C.; del Álamo, 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.
26. Keuning, H.; Jeuring, J.; Heeren, B. Towards a systematic review of automated feedback generation for programming exercises. In Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, Arequipa, Peru, 11–13 July 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 41–46.
27. 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]
28. Lajis, A.; Baharudin, S.A.; Ab Kadir, D.; 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. (JTEC)* **2018**, *10*, 109–113.
29. Pieterse, V. Automated Assessment of Programming Assignments. In Proceedings of the 3rd Computer Science Education Research Conference (CSERC '13), Heerlen, The Netherlands, 4–5 April 2013; pp. 45–56.
30. Leal, J.P.; Silva, F. Mooshak: A Web-based multi-site programming contest system. *Softw. Pract. Exp.* **2003**, *33*, 567–581. [CrossRef]
31. Eldering, J.; Gerritsen, N.; Johnson, K.; Kinkhorst, T.; Werth, T. DOMjudge. Available online: https://www.domjudge.org (accessed on 24 August 2020).
32. Joy, M.; Griffiths, N.; Boyatt, R. The BOSS Online Submission and Assessment System. *ACM J. Educ. Resour. Comput.* **2005**, *5*, 2. [CrossRef]
33. Gotel, O.; Scharff, C. Adapting an open-source web-based assessment system for the automated assessment of programming problems. In Proceedings of the Sixth Conference on IASTED International Conference Web-Based Education, Anaheim, CA, USA, 14–16 March 2007; Volume 2, pp. 437–442.
34. Srikant, S.; Aggarwal, V. A system to grade computer programming skills using machine learning. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 23–27 August 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 1887–1896.
35. Chakraverty, S.; Chakraborty, P. Tools and Techniques for Teaching Computer Programming: A Review. *J. Educ. Technol. Syst.* **2020**. [CrossRef]
36. Bai, X. Enhancing the learning process in programming courses through an automated feedback and assignment management system. *Issues Inf. Syst.* **2016**, *17*, 165–175.
37. Amelung, M.; Krieger, K.; Rösner, D. E-assessment as a service. *IEEE Transact. Learn. Technol.* **2011**, *4*, 162–174. [CrossRef]
38. Shute, V.J. Focus on Formative Feedback. *Rev. Educ. Res.* **2008**, *78*, 153–189. [CrossRef]
39. Gaudencio, M.; Dantas, A.; Guerrero, D.D.S. Can computers compare student code solutions as well as teachers? In Proceedings of the 5th ACM Technical Symposium on Computer Science Education—SIGCSE, Atlanta, GA, USA, 5–8 March 2014; Association for Computing Machinery: New York, NY, USA; pp. 21–26.
40. Jurado, F.; Redondo, M.A.; Ortega, M. Using fuzzy logic applied to software metrics and test cases to assess programming assignments and give advice. *J. Netw. Comput. Appl.* **2012**, *35*, 695–712. [CrossRef]
41. Singh, R.; Gulwani, S.; Solar-Lezama, A. Automated feedback generation for introductory programming assignments. In Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation—PLDI '13, Seatle, WA, USA, 16–22 June 2013; Association for Computing Machinery (ACM): New York, NY, USA, 2013; p. 15.
42. 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 Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2006, Bologna, Italy, 26–28 June 2006; Association for Computing Machinery (ACM): Bologna, Italy, 2006; p. 13.

43. Edwards, S.H. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.* **2003**, *3*, 1. [CrossRef]
44. 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]
45. 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]
46. Restrepo-Calle, F.; Ramírez Echeverry, 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]
47. Gordillo, A. Effect of an Instructor-Centered Tool for Automatic Assessment of Programming Assignments on Students' Perceptions and Performance. *Sustainability* **2019**, *11*, 5568. [CrossRef]
48. Wang, T.; Su, X.; Ma, P.; Wang, Y.; Wang, K. Ability-training-oriented automated assessment in introductory programming course. *Compu. Educ.* **2011**, *56*, 220–226. [CrossRef]
49. Gárcia-Mateos, G.; Fernández-Alemán, J.L. A course on algorithms and data structures using on-line judging. In Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE, Paris, France, 6–9 July 2009; ACM Press: New York, NY, USA, 2009; pp. 45–49.
50. 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]
51. Yin, R.K. *Case Study Research: Design and Methods*, 5th ed.; SAGE Publications, Inc.: London, UK; Thousand Oaks, CA, USA, 2014; ISBN 1452242569.
52. Quemada, J.; Barra, E.; Gordillo, A.; Pavon, S.; Salvachua, J.; Vazquez, I.; López-Pernas, S. Ammil: A methodology for developing video-based learning courses. In Proceedings of the ICERI2019 Proceedings, Seville, Spain, 11–13 November 2019; pp. 4893–4901.
53. Richard, J. *Light Making the Most of College: Students Speak Their Minds*; Harvard University Press: Cambridge, MA, USA, 2001; ISBN 9780674004788.
54. Edwards, S.H. Using Test-Driven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance. In Proceedings of the international conference on education and information systems: Technologies and applications EISTA, Orlando, FL, USA, 31 July–2 August 2003.
55. Epstein, M.L.; Lazarus, A.D.; Calvano, T.B.; Matthews, K.A.; Hendel, R.A.; Epstein, B.B.; Brosvic, G.M. Immediate feedback assessment technique promotes learning and corrects inaccurate first responses. *Psychol. Rec.* **2002**, *52*, 187–201. [CrossRef]
56. Fu, X.; Peltsverger, B.; Qian, K.; Liu, J.; Tao, L. APOGEE-Automated Project Grading and Instant Feedback System for Web Based Computing. In Proceedings of the 39th ACM Technical Symposium on Computer Science Education SIGCSE'08, New York, NY, USA, 12–15 March 2008; pp. 77–81.
57. Gulikers, J.T.M.; Bastiaens, T.J.; Kirschner, P.A.; Kester, L. Authenticity is in the eye of the beholder: Student and teacher perceptions of assessment authenticity. *J. Vocat. Educ. Train.* **2008**, *60*, 401–412. [CrossRef]
58. Gulikers, J.; Bastiaens, T.; Kirschner, P. Authentic assessment, student and teacher perceptions: The practical value of the five-dimensional framework. *J. Vocat. Educ. Train.* **2006**, *58*, 337–357. [CrossRef]
59. Gibbs, G.; Simpson, C. Conditions Under Which Assessment Supports Students' Learning. *Learn. Teach. High. Educ.* **2005**, *1*, 3–31.
60. Higgins, R.; Hartley, P.; Skelton, A. The conscientious consumer: Reconsidering the role of assessment feedback in student learning. *Stud. High. Educ.* **2002**, *27*, 53–64. [CrossRef]
61. Rößling, G.; Joy, M.; Moreno, A.; Radenski, A.; Malmi, L.; Kerren, A.; Naps, T.; Ross, R.J.; Clancy, M.; Korhonen, A.; et al. Enhancing learning management systems to better support computer science education. *ACM SIGCSE Bull.* **2008**, *40*, 142–166. [CrossRef]
62. Glitch. Available online: https://glitch.com/ (accessed on 24 August 2020).
63. GitHub. Available online: https://github.com (accessed on 4 August 2020).
64. Moodle API. Available online: https://docs.moodle.org/dev/Core_APIs (accessed on 4 August 2020).
65. Mocha. Available online: https://mochajs.org/ (accessed on 4 August 2020).
66. ZombieJS. Available online: http://zombie.js.org/ (accessed on 4 August 2020).
67. Chai. Available online: https://www.chaijs.com/ (accessed on 4 August 2020).
68. Watson, C.; Li, F.W. Failure rates in introductory programming revisited. In Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education ITiCSE '14, Uppsala, Sweden, 21–25 June 2014; pp. 39–44.

69. Bandura, A. Self-efficacy mechanism in human agency. *Am. Psychol.* **1982**, *37*, 122–147. [CrossRef]
70. Bloom, B.S. *Taxonomy of Educational Objectives: The Classification of Education Goals by a Committee of College and University Examiners*; David McKay: Philadelphia, PA, USA, 1956.
71. Chory-Assad, R.M. Classroom justice: Perceptions of fairness as a predictor of student motivation, learning, and aggression. *Commun. Q.* **2002**, *50*, 58–77. [CrossRef]
72. Ministerio de Universidades. *Recomendaciones del Ministerio de Universidades a la Comunidad Universitaria para Adaptar el Curso Universitario 2020–2021 a una Presencialidad Adaptada*; Ministerio de Universidades: Madrid, Spain, 2020. Available online: https://www.ciencia.gob.es/stfls/MICINN/Universidades/Ficheros/Recomendaciones_del_Ministerio_de_Universidades_para_adaptar_curso.pdf (accessed on 4 August 2020).