MAESTRO: a semi-autoMAted Evaluation SysTem for pROgramming assignments

Alessandro Bertagnon
Department of Engineering
Ferrara University
Ferrara, Italy
https://orcid.org/0000-0003-2390-0629
alessandro.bertagnon@unife.it

Marco Gavanelli
Department of Engineering
Ferrara University
Ferrara, Italy
https://orcid.org/0000-0001-7433-5899
marco.gavanelli@unife.it

***Abstract*—Many works in the literature highlight the importance of formative assessment to improve the learning process. Formative assessment means providing assignments to the students and giving them feedback during the course. It is not very used in Italian Universities because it is highly time-consuming for the professors, that typically do not have help in the process of homework grading.**

In this work, we focus on programming exercises in computer science subjects, and propose a tool to semi-automatically grade and give feedback to the students. The tool was used in a computer language course on functional programming in a M.Sc. degree; the students evaluation of the course show a steep increase in the students appreciation. The tool is currently used in a course at undergraduate level on C programming.

***Index Terms*—Distance learning methods and technologies; Computer and web-based software for instruction; Debugging tools and learning; software engineering programming issues and laboratory practice.**

I. Introduction

The effectiveness of *formative assessment* has been pointed out in many works, e.g., in the classical survey by Black and Wiliam [1]. The term Formative Assessment, opposed to Summative Assessment, was proposed for the first time in [2] with its current meaning. While Summative Assessment is performed at the end of a course, teaching unit or semester with the aim of grading or certification (of the student, of the teacher or of the curriculum) formative assessment is used during the process of teaching and learning with the purpose of improving the teaching and/or the learning process.

In the Italian higher-education system, usually the only evaluation that students have is during the final examination, and as such falls into the summative assessment category. This has a series of drawbacks. One is that assessment is done when it is too late to provide any constructive feedback to the students, and if a student misunderstands a concept early in the semester, it is not immediately corrected, possibly making fruitless vast parts of the following efforts of the student. Also, the exam is a very stressful situation and students might be tempted to cheat; this causes that the teacher does not get a real understanding of the level of achievement of the objectives, since the only assessment is through the exam.

Even when professors understand that students learning could be improved if they had feedback on their exercises, this is considered inapplicable in practice, because there is not enough personnel to correct students' exercises. Quizzes (e.g., with yes/no or multiple choice questions) can be corrected with automatic tools, but they are often quickly ruled out: many professors think that in their subject quizzes would not be suitable, since the result of the exam is not a yes/no answer or a numeric value, but the methodology to obtain a result. For example, when learning a programming language, the exam often comprises tests in which students write a program, and it is difficult (if not impossible) to devise yes/no or multiple choice questions to assess the programming skills of a student.

Usually, the main interaction between students and professor happens during lectures, where students can raise their hand to ask questions, in case they did not understand a passage. On the other hand, this happens quite rarely, often only the most self-confident students ask for explanations, while shy students tend to remain silent, because they fear judgment from the professor of from the other students in the classroom.

This became even worse in the second semester of 2019, when Italian universities had to adopt distance learning, due to the Coronavirus disease 2019 (COVID-19) lockdown. Many professors started giving lessons in streaming, and they complained that they could not get feedback from the students, since they could not even see the faces of their students.

In Italian Universities, on average students take 41% more time to graduate with respect to the expected duration of the studies; this index has wide variations depending on the subject of studies, and it is 51% in Engineering [3].

In this work, we propose semi-autoMAted Evaluation SysTem for pROgramming assignments (MAESTRO), a semi-automated tool to check students' programs, and to give feedback to the students. The idea is to ask students to upload their computer programs, and check through software engineering technologies (in particular, by running tests) the correctness of the students' programs; in case the tool detects an error, this is communicated to the student, together with the test case that failed. The tool was developed and then

We thank Prof. Evelina Lamma for her help in this research. We thank Elisa Gulmini for helping with retrieving some of the data. We thank Alessandro Zerbinati for the implementation of the C language part.
This work was partially supported by GNCS-INdAM.

used in the course *Programming Languages and Compilers* (Linguaggi e Traduttori) in Ferrara University (Italy).

The rest of the paper is organized as follows. In next section, we describe the main case study: the course Programming Languages and Compilers. We then explain, in Section III how the course was organized in 2019, during the lockdown due to COVID-19. Section IV explains the software technologies underlying MAESTRO, and its architecture. We provide the results obtained with MAESTRO in Section V. The tool is currently used in a second case study, for a first-year course on computer science; we report on its current use in Section VI. Finally, we propose extensions for future work and conclude.

II. Case study: the course "Programming Languages and Compilers"

As a case study, we will consider the course *Programming Languages and Compilers* (Linguaggi e Traduttori) in the Laurea Magistrale (equivalent to the M.Sc. level) *Computer Science and Automation Engineering* (Ingegneria Informatica e dell'Automazione). The course is held once every two years, and is attended by about 50 students in each edition.

The course Programming Languages and Compilers insists on two main subjects: parsers and functional programming. In particular, in functional programming students learn how to program in the Haskell language exploiting the main features of the language: pure declarativity, higher-order functions, lazy evaluation, monads. Haskell is a strongly typed language, and Haskell compilers detect, at compile time, many errors that in other languages would occur only at run-time. Students know imperative and object-oriented programming from their *laurea* courses (roughly equivalent to B.Sc.), and have difficulties in understanding the functional programming paradigm, since it requires a completely new way of thinking about programs.

For this reason, the course contains a part held in the computer laboratory of about 15 hours, out of the 60 hours of lessons. In the lab lessons, students are assigned exercises, and they can raise their hand to ask for help to the professor. The professor walks around the lab, asks students if they were able to solve the problems, and possibly has a look at computer monitors, providing some suggestions. Students works are not graded, and students do not get any feedback on their work (unless they raise their hand). This is a rather classical organization of laboratory lessons in Italian Universities.

III. The new organization of the course

The 2020 edition started immediately after the lockdown due to the COVID-19 pandemic: lessons could not be given in the classroom (or in the lab), but only on-line. This required a complete rethinking of the organization of the lessons, in particular to foster interaction with the students in a situation where the only communication possible was through the Internet.

The lessons were pre-recorded videos where the professor explained the various topics; about once every one or two weeks, there was a synchronous session where students could discuss with the professor. However, the main fear was that in these synchronous sessions students would not ask questions, due, e.g., to shyness.

To raise the level of interaction, and also have a valid substitute for the laboratory part, the idea was to introduce formative assessment: assign students exercises and give them feedback during the course. Since the formative objective of the functional programming part was to have students learn a new programming paradigm, the natural means was to assign exercises that students had to solve through Haskell programs. The professor would collect the programs, correct the assignments and provide a grade, together with a personal feedback to each student. The synchronous sessions would then be used to discuss frequent (or, significant) errors with the students, while, obviously, hiding the name of the student that made each error. Since students would have already faced the difficulties of the exercise, they would be interested to know how to reason in that specific case.

On the other hand, although this organization seemed promising, it raised the problem of correcting a large number of exercises: only one person was available to correct the assignments, namely the professor, and of course university professors also have other duties beside teaching.

To reduce the time spent in correcting programs, a computer-assisted correction of the assignments was devised. Of course, in general it is impossible to have a completely automated procedure to check the correctness of programs in a Turing-complete language (such as Haskell), since even checking if a program loops is not computable [4]. On the other hand, even an incomplete procedure would be useful, to shortlist the programs to be manually corrected.

In next section, we explain the technologies used in the Programming Languages and Compilers course for the semi-automatic correction of students' programs, and the architecture of the application.

IV. Software technologies and architecture of MAESTRO

A. Google Classroom, Google Forms

Google Classroom (GC)[1] is a web service developed by Google specifically for educational purposes, created with the aim of digitizing and simplifying the teaching workflow. Students in GC are divided into classes, in the university context each class corresponds to one course, and for each class there are tools that allow teachers and students to interact and share teaching material. GC includes the possibility to provide assignments to students to verify their skills and return feedback and assessments. Assignments management can be more or less automated and two different approaches can be substantially distinguished. In the first approach the student uploads the solution onto the service, the teacher reviews it and submits the assessment. In the second approach the grading is done by the service. This second approach is only possible when dealing with assignments in the form of quizzes where questions can be easily verified by direct comparison with the

[1]https://classroom.google.com

solution (e.g. multiple choice questions). GC quizzes support is made possible thanks to the integration with Google Forms. Google Forms (GF)[2] is a web app that is part of Google's office suite (now known as Google Workspace) and specifically designed to collect information. Due to its versatility, the service is currently used for a wide variety of purposes, including: event registration, surveys, data collection, quizzes, scheduling appointments, etc. The flexibility of the forms comes from the possibility, during their creation, to choose between many types of fields (short text answer, text field, multiple choice, checkboxes, dropdown, multiple choice grid, checkbox grid, etc) and to add multimedia content such as images, videos and URLs of external websites. There are also more advanced features such as the possibility to modify the flow of questions through conditional tools that change according to the user's choices. The data collected through the forms can be analysed and transformed into relevant information either with the analysis tools made available by Google Workspace or with external software. The data can be exported in various file formats (e.g. comma-separated values) recognized by spreadsheet and data analysis software. From a didactic perspective one of the most interesting features, introduced in 2016, is the ability to automatically grade quizzes. As already indicated above, it is possible to insert the answer keys when creating the form so that when the form is filled in, the grade is automatically computed, although this functionality is limited only to certain types of form fields.

B. Flubaroo

Flubaroo[3] is a Google Sheets add-on, launched in 2011, designed to evaluate and share the results of student assignments. The tool, which counts today more than 10 million installations and is used to evaluate more than 400000 assignments every month, is completely free to use and the source code is available on GitHub[4] under Apache-2.0 License. Although Flubaroo is an Add-on for Google Sheets, it is designed to work mainly on data collected through a Google Form. Flubaroo is simple to use. The usual way to work with the tool is to begin creating a Google Form with the assignment questions, and some fields to identify the student. Each time the form is filled in, the answers given by the students will be automatically stored in a linked Google spreadsheet. Once the teacher has entered the correct answers that will be used as answer keys and all the students have submitted their answers, it is possible to move on to the grading phase. For all questions in which there is a unique correct answer (or for which there is a list of correct answers), Flubaroo is able to compute the grade. The other questions must be graded by hand. As last step the teacher can review the grades, add some feedback for each student, and decide to email each student his/her grade. Flubaroo also provides important statistics for teaching such as: average assignment score, average score per question, low scoring questions and low scoring students. This data provides important insights concerning student competence and the quality of teaching. Many of Flubaroo's features have been introduced into Google Forms which, as mentioned above, now allows to automatically grade quizzes. However, Flubaroo is more flexible as it allows teachers to work on data extracted from forms and possibly pre-processed by other applications.

[2]https://docs.google.com/forms/

[3]http://www.flubaroo.com

[4]https://github.com/edcodeorg/flubaroo

C. QuickCheck

As already said, in general checking if a computer program is correct is not computable. On the other hand, tools that detect errors are widely used, because even detecting some errors saves precious human time.

Different tools have been proposed in practice to detect errors. Static analysis is one such options, and many such techniques are currently enclosed in modern compilers. Another option is to generate test cases and check if the program provides correct answers. Both approaches could be used for the task of semi-automatic checking of students' programs, we currently used only the second, and we leave or future work the exploitation of other techniques.

QuickCheck [5], [6] is probably the most famous tool of Property-Based Testing. Consider a function whose correctness should be tested. The programmer defines a *property* that should always be satisfied; this is given as a predicate, or a function returning a Boolean. Given the function to be tested and the property, QuickCheck generates a large (configurable) number of random tests, invokes the tested function for each test case, and checks that in all cases the property is satisfied. If some test case fails, it is reported to the user as a counterexample.

QuickCheck was designed in Haskell, but the idea was so successful that ports have been independently developed for most mainstream programming languages (the Wikipedia page lists dozens of them).

D. Architecture

Programming assignments are not trivial to evaluate because in order to verify their correctness it is necessary to resort to several software engineering technologies. The computer-assisted correction tools offered by cloud teaching services, such as those presented above, are unable to perform their task when it comes to evaluating programming assignments. For this reason we decided to develop a dedicated evaluation system that could be easily integrated into the workflow of the Google Workspace suite. In the proposed architecture (Figure 1) students' computer programs are collected through a Google Form, shared via Google Classroom, where students enter their identification data (name, surname, email) and the source code as text. The data of all assignments are then downloaded in the form of CSV file and uploaded into MAESTRO together with the source file containing a valid solution for the problem in question. The property tested with QuickCheck is typically that the result of the function written by the student and of the version written by the professor are the same; of course other properties may be used as well. MAESTRO, for each record of the CSV file provided as input,

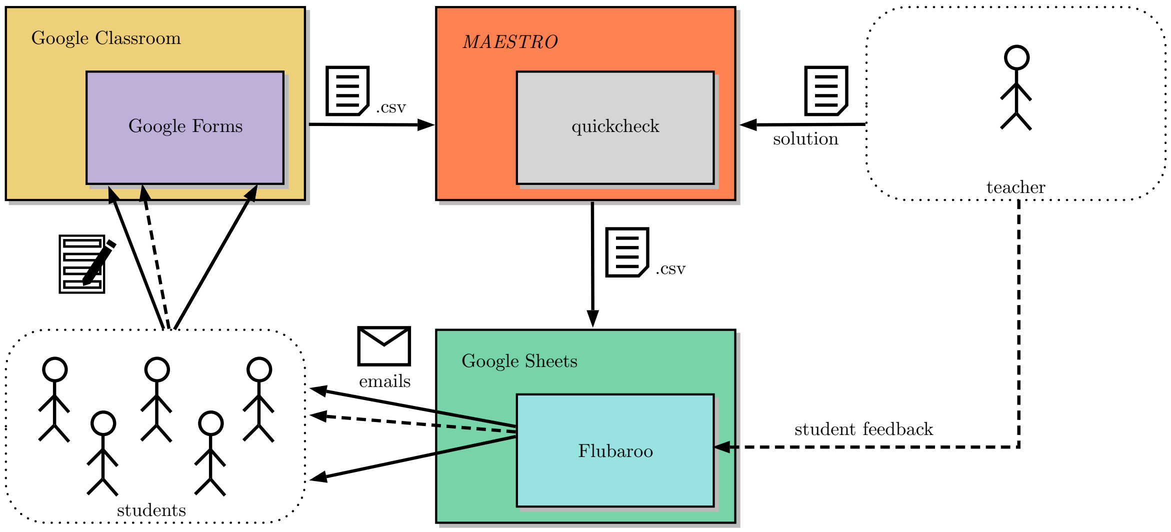

Fig. 1. Architecture

is able to perform several evaluation tasks on the student's source code including:

- check whether the requirements indicated in the assignment have been observed (e.g. function names and parameter types);
- check if the source code is compiled properly, by running the compiler, and if necessary report warnings and errors;
- execute the program and verify that it completes within an acceptable time considering the task it is carrying out;
- check for runtime errors;
- perform property-based testing through QuickCheck.

At the end of all the tasks, a specific report is generated for the student both in the form of text and in the form of a question-answer summary compatible with Flubaroo. MAESTRO exports the results for all the students as a CSV file. Table I reports the CSV fields generated by MAESTRO. Some fields allow multiple choice values, this makes them suitable for the computer-assisted grading. Values considered correct for each field are shown in bold; ND means "Not defined" and applies when the value cannot be evaluated (e.g. a program cannot be executed if the compilation fails). The `Text` fields add additional information useful to both the teacher and the student.

The teacher then re-uploads the generated CSV file to Google Sheets, and proceeds to grade each assignment using Flubaroo. Each CSV field, generated by MAESTRO, that supports computer-assisted grading is associated with a score. Flubaroo basically performs the sum of the scores of the fields with the correct value to compute the student's overall grade. The grade obtained by each student and the statistics of the class are provided in a separate spreadsheet. Finally, the teacher may decide to enter customised feedback for all, or just a few, students before sending each one the summary assessment email.

Field	Value
Was the code written according to specs?	[**Yes**, No]
Are errors generated during compilation?	[Yes, **No**]
Are warnings generated during compilation?	[Yes, **No**]
Did runtime errors occur?	[Yes, **No**, ND]
Did the program end in an acceptable amount of time?	[**Yes**, No, ND]
Is the output provided according to specs?	[**Yes**, No, ND]
Are the results computed correctly?	[**Yes**, No, ND]
Compiler output	`Text`
Execution output	`Text`
Testing output	`Text`

TABLE I
CSV FIELDS GENERATED BY MAESTRO.

V. RESULTS

The statistics of student evaluation of courses for Ferrara University are published on the SIS-ValDidat system [7], which is used by 20 Universities in Italy. The results are publicly available on the web[5] (except for the rankings, which are available only upon authentication). Table II reports the results of the last edition before the introduction of formative assessment, while Table III contains the same results after that introduction. The evaluation for each question is given on the usual scoring system in use in Italy, in which marks are on the 0-10 scale, and the pass mark for exams is 6 out of 10. The tables provide the average mark obtained in the question, the standard deviation, the average mark obtained by courses in the degree course, and the rank obtained by Programming Languages and Compilers out of the courses in the degree. Note that the SIS-ValDidat system does not report a score if less than 5 students answered that specific question, or if for a course there were less than 5 answers overall. This explains also the fact that the number of courses in the *"rank"* column has some variability. After the lockdown due to the COVID-19

[5]http://valmon.disia.unifi.it/sisvaldidat/unife/

Avg	Std.Dev.	Degree course	Rank	Question
8.04	2.289	8.3	11/ 18	Was preliminary knowledge enough to understand the subject?
6.64	3.097	8.19	17/ 18	Is the workload of this course proportionate to the number of credits?
8	2.191	8.26	13/ 18	Is the teaching material adequate?
8.76	1.727	8.93	14/ 18	Have the methods of examination been clearly defined?
9.63	0.992	9.23	5/ 17	Are teaching hours respected?
8.17	2.249	8.33	11/ 17	Does the professor stimulate / motivate interest?
8.92	1.935	8.34	6/ 17	Does the professor explain the arguments clearly?
8.13	1.727	7.91	9/ 17	Are exercises, laboratories, etc. useful?
9.13	1.364	9.08	8/ 17	Are the subjects consistent with what is stated on the website?
9.28	1.281	8.71	5/ 18	Is the professor actually available for clarifications and explanations?
7.6	2.498	8.39	14/ 18	Are you interested in the subject?

TABLE II
RESULTS OF THE STUDENTS EVALUATION OF THE COURSE *Programming Languages and Compilers* IN THE LAST EDITION BEFORE THE INTRODUCTION OF FORMATIVE ASSESSMENT THROUGH SEMI-AUTOMATIC EVALUATION OF PROGRAMMING EXERCISES.

Avg	Std.Dev.	Degree course	Rank	Question
8.94	1.779	7.95	1/18	Was preliminary knowledge enough to understand the subject?
7.72	1.967	7.55	7/18	Is the workload of this course proportionate to the number of credits?
8.89	1.629	7.81	1/18	Is the teaching material adequate?
8.44	1.833	8.36	9/18	Have the methods of examination been clearly defined?
-	-	-	-	Are teaching hours respected?
9.03	1.5	8.11	2/17	Does the professor stimulate / motivate interest?
9.33	1.247	8.21	2/17	Does the professor explain the arguments clearly?
8.95	1.581	7.86	4/15	Are exercises, laboratories, etc. useful?
9	1.414	8.95	6/17	Are the subjects consistent with what is stated on the website?
9.42	1.187	8.71	2/18	Is the professor actually available for clarifications and explanations?
8.17	1.724	8.01	7/18	Are you interested in the subject?
8.94	1.527	7.78	1/8	Overall, do you think the online teaching is effective?

TABLE III
RESULTS OF THE STUDENTS EVALUATION OF THE COURSE *Programming Languages and Compilers* AFTER THE INTRODUCTION OF FORMATIVE ASSESSMENT THROUGH SEMI-AUTOMATIC EVALUATION OF PROGRAMMING EXERCISES.

disease, a new question was introduced, about the effectiveness of the online teaching. This question was introduced only for the courses on the second semester, about half of the total courses in the degree.

Before the introduction of MAESTRO (Table II), Programming Languages and Compilers got a pass mark in all questions, although the score was below the average of the courses in the degree on 6 questions (preliminary knowledge, workload, teaching material, methods of examination, motivation of the professor, and interest of the students). The workload was considered particularly high, and Programming Languages and Compilers ranked second-last of all courses in the degree on this criterion.

After the introduction of the semi-automatic evaluation (Table III), there was an abrupt improvement of the scores in almost all questions. Before the introduction of MAESTRO, Programming Languages and Compilers ranked between the 5th and the 17th place (out of 18), while after its introduction it ranked between the 1st and the 9th place (out of 18). In particular, it was considered the best course according to the effectiveness of on-line teaching.

The percentage of students that passed the exam in the first six months after the end of the lessons also increased: from 65.6% to 72.5%.

Also, in the previous edition, 15.6% of the students who had chosen Programming Languages and Compilers in their study plan, decided to substitute it with other exams, that were considered easier by the students. After the introduction of MAESTRO, none of the students changed their study plan to remove Programming Languages and Compilers.

The synchronous sessions were very lively, and the professor was able to discuss the errors made by the students.

Some lessons were learned from the experience. If the assignments are very easy, then most of the students do it right, and the time spent for correcting the few errors is low. This means that the focus is on lower-level students, while higher level students just get as feedback that their work is correct, but do not get any correction to improve their skills. On the other hand, lower-level students are the ones most needing help, so this choice is not necessarily wrong. If, instead, the exercises are harder, then the time to explain the errors becomes higher, with the risk to overwhelm the professor.

VI. WORK IN PROGRESS: THE COURSE "BASICS OF COMPUTER SCIENCE AND LABORATORY"

As a second case-study, we have re-implemented MAESTRO to use it for language C, and it is currently used to automatically correct students' programs in a computer science basics course: the course *Basics of Computer Science and Laboratory* (Fondamenti di Informatica e laboratorio).

There are currently 258 students that subscribed to the Google Classroom of the course, although not all of them do their homework. Each week, an assignment is given to them, and they can submit their solution; it will be graded in the following week. In the first assignment, it took about 130 minutes to correct 89 works. Most of the time was devoted to write comments for the students: first year students have difficulties even in understanding the error messages of the compiler, so they need detailed explanations of why their program could not be compiled.

In most cases, programs that passed all the tests were considered correct. Of course, this is not always precise, and we selected randomly a small number of programs that passed the tests for a deeper hand-made analysis. No errors were discovered in this way.

Other simple tests were added as spreadsheet formulas, e.g., testing if forbidden structures were used in the program (e.g., `goto` statements or other assignment-based restrictions).

VII. Related Work

Many tools exist for automated assessment of computer science assignments; the excellent survey [8] provides various pointers. The most closely related work is probably Ceilidh [9] and its successor CourseMarker [10]. CourseMarker is written in Java and executes all the tests on the students' computers. The tests to be performed are written by the teacher. Web-CAT [11] is another highly customizable open source automated grading system that supports many programming languages. Unlike other systems, Web-CAT requires students to write tests for their own code. GradeIT [12] uses program repair techniques to evaluate also programs that do not compile and therefore provide evaluations more similar to those provided by teaching assistants.

These are large systems developed over several years that embed user interfaces for both students and teachers, several types of users, customizations. MAESTRO is a much simpler application, it relies on Google Forms (and, as such, it is easy to use for the students that are already familiar with Google Classroom), and uses QuickCheck to automatically execute a large number of random tests.

VIII. Conclusions and Future Work

In this paper, we presented MAESTRO, a tool for semi-automatic assessment of students programs. It was used in a university course on programming languages to automatically provide feedback on students' assignments (formative assessment). The students' feedback was enthusiastic: many of them congratulated with the professor after the exam, and the students evaluation of the course improved significantly (from a less-than-average course among the ones in the degree, to a top-ranked course in various criteria). The tool is currently being used also for a computer science basic course.

The long-term objective of this work is to have an application that is completely web-based, without resorting to downloading files and re-uploading results. Such an application could integrate with Google Classroom and/or Flubaroo, and provide the feedback immediately after the submission of the works by the students. We foresee that some user intervention could be necessary in some cases, but the main objective is to automatize as much as possible the process, in order to automatically grade a large number of works, while leaving to the professor the decision on edge cases.

The number of tests could be enlarged, in order to include other software engineering technologies, such as static analysis, concolic testing, or different testing methodologies.

Also, for simple programs it is likely that many submitted programs represent exactly the same algorithm, although possibly written in syntactically different ways (e.g., different variable names, swapping of instructions that do not require a specific ordering, etc.). By grouping programs that are actually the same solution, the professor time to correct programs could be further reduced, while providing the same answer to many students. Of course, in general detecting if two programs are the same is not computable in general, but even detecting in some cases the equality of programs would reduce the overall human effort. This could be done by comparing the Abstract Syntax Trees of the students' programs. There exists similarity testing programs, but they are usually focussed on different tasks, such as detecting plagiarism [13].

References

[1] P. Black and D. Wiliam, "Assessment and classroom learning," *Assessment in Education: Principles Policy and Practice*, vol. 5, no. 1, pp. 7–73, 1998.

[2] B. S. Bloom, J. T. Hastings, G. F. Madaus, and T. S. Baldwin, Eds., *Handbook on the formative and summative evaluation of student learning*. New York, NY: McGraw-Hill, 1971.

[3] Consorzio Interuniversitario AlmaLaurea, "XXII indagine - profilo dei laureati 2019," 2020.

[4] A. M. Turing, "On computable numbers, with an application to the entscheidungsproblem," in *Proceedings of the London Mathematical Society*, ser. 2, vol. 42, 1937, pp. 230–265.

[5] K. Claessen and J. Hughes, "QuickCheck: a lightweight tool for random testing of Haskell programs," in *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, M. Odersky and P. Wadler, Eds. ACM, 2000, pp. 268–279.

[6] J. Hughes, "QuickCheck testing for fun and profit," in *Practical Aspects of Declarative Languages, 9th International Symposium, PADL 2007, Nice, France, January 14-15, 2007*, ser. Lecture Notes in Computer Science, M. Hanus, Ed., vol. 4354. Springer, 2007, pp. 1–32.

[7] B. Chiandotto and B. Bertaccini, "SIS-ValDidat: a statistical information system for evaluating university teaching," *Quaderni Di Statistica*, vol. 10, pp. 157–176, 2008.

[8] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, pp. 83 – 102, June 2005.

[9] E. Foxley, C. Higgins, A. Tsintsifas, and P. Symeonidis, "Ceilidh: A system for the automatic evaluation of student programming work," in *Proceedings of the 4th International Conference on Computer Based Learning in Science (CBLIS'99)*, 1999.

[10] E. Foxley, C. Higgins, T. Hegazy, P. Symeonidis, and A. Tsintsifas, "The CourseMarker CBA system: Improvements over Ceilidh," in *Proc. of the 5th Annual Computer Assisted Assessment Conference*, 2001.

[11] S. H. Edwards and M. A. Perez-Quinones, "Web-cat: Automatically grading programming assignments," *SIGCSE Bull.*, vol. 40, no. 3, p. 328, Jun. 2008. [Online]. Available: https://doi.org/10.1145/1597849.1384371

[12] S. Parihar, Z. Dadachanji, P. K. Singh, R. Das, A. Karkare, and A. Bhattacharya, "Automatic grading and feedback using program repair for introductory programming courses," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '17. ACM, 2017.

[13] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *Proc. of the 2003 ACM SIGMOD international conference on Management of data*, 2003.