Received July 27, 2019, accepted August 22, 2019, date of publication August 29, 2019, date of current version September 24, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2938391

Automated Assessment of Computer Programming Practices: The 8-Years UNED Experience

DANIEL GALAN[1], RUBEN HERADIO[2], HECTOR VARGAS[3], ISMAEL ABAD[2], AND JOSE A. CERRADA[2]

[1]Department of Computer Science and Automatic Control, Universidad Nacional de Educacion a Distancia (UNED), 28040 Madrid, Spain
[2]Department of Software and Systems Engineering, Universidad Nacional de Educacion a Distancia (UNED), 28040 Madrid, Spain
[3]School of Electrical Engineering, Pontificia Universidad Catolica de Valparaiso, Valparaiso 2374631, Chile

Corresponding author: Daniel Galan (dgalan@dia.uned.es)

This work was supported in part by the Spanish Ministry of Science, Innovation and Universities, under Project DPI2016-77677-P and Project DPI2017-84259-C2-2-R, and in part by the Community of Madrid under Grant RoboCity2030-DIH-CM P2018/NMT-4331.

ABSTRACT The increasing popularity of distance education poses exciting new challenges. In particular, current pedagogical paradigms, such as competency-based education, require students' continuous evaluation. That is, to master skills, students need to receive constant feedback to guide their experimentation processes. However, teaching teams are usually under-dimensioned to support the large number of students that online courses usually have. This paper presents the approach we have adopted at the National University of Distance Education to overcome this problem for the case of computer programming practices, which complements human evaluation with an automatic assessment system. The paper describes our system and reports its benefits with an empirical study from 2011 to 2018 that involved 14,944 students.

INDEX TERMS Distance education, online education, automated assessment, computer programming training.

I. INTRODUCTION

Open and Distance Learning (ODL) has become a fundamental educational pillar, promoting social and economic development [1]. The amount of people taking Massive Open Online Courses (MOOCs) or enrolling in Distance Learning Universities is growing year by year, even though the overall number of students registered in higher education worldwide has decreased [2]. For instance, two of the most prestigious European distance universities, the British Open University and the Spanish National University of Distance Education (UNED) had 174,898 [3] and 165,855 [4] students in the academic course 2018-19, respectively. In the case of MOOCs, these figures are even more impressive: Coursera has more than 40 million students [5], edX 20 million [6], and XuetangX 14 million [7].

Accordingly, ODL institutions face a critical challenge: how to support an enormous volume of students without limiting the educational quality of their courses. For example, it is beyond any doubt that computer science students benefit notably from performing programming exercises and receiving the corresponding instructors' feedback [8], [9]. However, the number of practices proposed for evaluation tends to be inversely proportional to the number of students [10]. A workload unmanageable by the teaching team leads to a deficit in the students' practical learning and, consequently, it may affect their academic performance negatively.

The associate editor coordinating the review of this manuscript and approving it for publication was Chin-Feng Lai.

This paper presents the approach we have applied to teach computer programming at UNED, reporting its successful pedagogical results. We have developed a framework to review programming assignments automatically. The framework receives two inputs from the instructor: (i) a program that correctly implements the assignment, and (ii) a set of input values for the program. Then, it applies *combinatorial testing* techniques [11] to combine those values for extensively checking if the student's and instructor's programs produce the same output. Whenever the comparison fails, some feedback is given to the student. Obviously, the framework can only verify students' program functionality, but not other program quality attributes of interest, such as its legibility, modularity, etc. Therefore, we follow a blended approach that combines both Automatic Assessment (AA) with human review.

The benefits of our mixed methodology have been tested over eight consecutive academic courses (from 2011 to 2018) with a total of 14,944 enrolled students (1,868 students per course on average), showing that our automatic system adequately prepares students to competently attempt the more complex assignments that will be corrected by humans, and that there is a strong correlation between the completion of practical assignments and (i) obtaining high exam grades, and (ii) preventing dropout, which is a critical problem in ODL (e.g., in the Open University, there has been reported a 78% students' dropout rate [12]).

The remainder of this paper is organized as it follows: Section II summarizes related work and highlights the contributions of this paper; Section III describes our AA framework; Section IV reports our methodology's empirical evaluation; Finally, Section V provides some concluding remarks.

II. RELATED WORK

According to [13], [14], the first AA system for computer programming assignments was developed by Hollingsworth [15] in 1960. This system evaluated assembly code *written* in punch cards. Since then, and particularly in the last years, many other AA systems have appeared: Automata [16], Dr. Scratch [17], EduPCR [18], an AA system for learning object-oriented programming [19], VPL [20], Grading Java Assigments [21], GradeIT [22], an AA for OpenGL [23], and so on.

Despite the large number of available AA systems, there are few literature reviews [24]–[27] that provide an overview of the field. Nevertheless, AA systems are usually classified into *dynamic*, *static*, and *hybrid*, according to the type of analysis they perform [28], [29]:

- **Dynamic analysis** is the most popular one. It involves executing the code varying the input parameters. Then the system compares the output with the one generated by a code that is known to be correct. If they match then the program is considered to pass the analysis. Many systems adopt this method, such as CodeMaster [30], the AA gamified approach [31], and Flexible Dynamic Analyzer [32].
- **Static analysis** is characterized by studying the code without executing it, that is, it focuses exclusively on the program structure. It informs about the degree of compliance with the specifications requested. The fact that a program is correct in its dynamic analysis (produces the expected results) does not mean that it is also correct statically (the programming structure is not adequate). Due to the complexity of this method, few projects implement it [14], [33], [34].
- **Hybrid analysis** combines both methods above to provide a complete code analysis. Hybrid systems are often the result of unifying previously developed analyses [35]–[37].

Concerning the pedagogical value of AA systems, in 2019, Restrepo-Calle et al. [38] reports an empirical validation examining the final students' marks along three different semesters. Other authors have focused their attention on how students' perceive and appreciate AA online tools [39], [40].

This paper presents a dynamic system that requires instructors a minimum effort: they only need to provide a valid version of the program students need to solve, and some input values for the program. In turn, it generates a combinatorial test suit that widely checks students programs. In contrast to most published empirical studies, that validate AA systems on small samples of tens of students over one academic course [17], [38]–[40], this paper reports the experience of thousands of students over eight courses.

III. A SOFTWARE TESTING APPROACH FOR AUTOMATED PRACTICE ASSESSMENT

This section presents our AA framework from a practical point of view: introducing a programming exercise that students' need to solve, discussing the rationale behind the input values instructors must provide, and showing how the framework combines those values to test students' answers.

A. RUNNING EXAMPLE

Imagine an assignment where students are requested to write a C++ program that prints on the screen the calendar sheet of any month and year between 1601 and 3000. The program first asks the user for the month and year to be shown, and then it displays a sheet like Listing 1, finishing its execution. The program should not print anything when the inputs are out of their range, and it must always end (i.e., indefinite loops are forbidden).

The following hints are given to the students to fulfill their assignment:

- January 1st, 1601 was Monday.
- Regular years have 365 days. Leap years have 366 days (February 29th is the extra day).
- A year is leap if it is multiple of 4, except if it can be exactly divided by 100. The exception does not apply when the year is also multiple of 400. For instance, 1604 is leap because it is divisible by 4, but not by 100. 1800 is not leap, because it is multiple of 4 and 100, but not of 400. Finally, 2000 is leap as it is divisible by 4, 100, and 400.

From a methodological point of view, students are instructed to use the *stepwise refinement* strategy [41], breaking down the problem into functions and procedures. They are encouraged to write a function to determine whether a year is a leap or not, and another to calculate the day of the week with which a month of a year begins. They are also advised to carry out an auxiliary procedure to print the days of the calendar

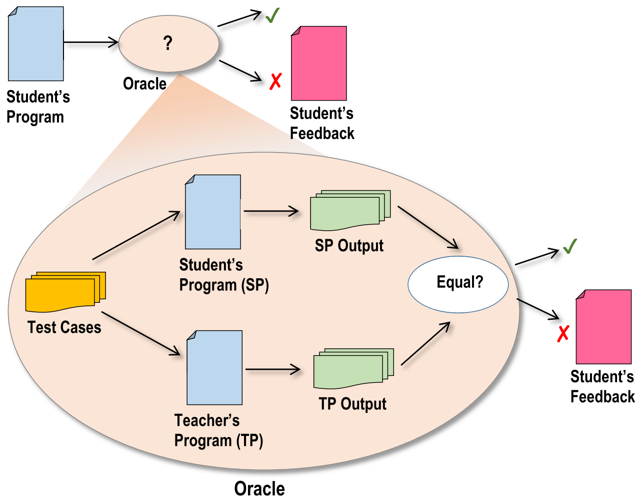

FIGURE 1. Differential testing approach for practice automated assessment.

```
Month (1..12)? 9
Year (1601..3000)? 2010

SEPTEMBER                   2010
===========================
MO  TU  WE  TH  FR | SA  SU
===========================
 .   .   1   2   3 |  4   5
 6   7   8   9  10 | 11  12
13  14  15  16  17 | 18  19
20  21  22  23  24 | 25  26
27  28  29  30   . |  .   .
```

Listing 1. Execution example of the calendar program .

sheet in the format requested. Finally, they are recommended to use the C++ `enum` type to handle the days of the week.

B. AA FRAMEWORK

Figure 1 sketches the operation of our framework. At a high abstraction level, it receives a student's program, checking its functional correctness. In *software testing* terminology, this kind of systems are called *oracles* [42]. In particular, our oracle is built under the *differential testing* approach [43]–[45], that is, the student's solution is compared to the instructor's one by running multiple test cases, and then checking if the two program outputs coincide (within a customizable precision range). Whenever a test case fails, the oracle gives feedback to the student, justifying why her/his program is not correct.

Guaranteeing that the student's program is absolutely correct would most times require an infinite test suite that accounts for every possible valid program input. In particular, testing the calendar program would require:

$$\begin{cases} [1601, 3000] \wedge [1, 12] & \Rightarrow 16,788 \text{ valid possibilities} \\ (-\infty, 1600] \vee [3001, +\infty) & \Rightarrow \infty \text{ invalid possibilities} \end{cases}$$

This problem has also been studied by the software testing community. Specifically, the *domain testing* approach [46] overcomes the endless test case problem using *equivalence classes*:

- The range of every input variable is broken into subsets conceptually tantamount, called equivalence classes, for the program under test.
- Only a few class members need to be selected for the test cases. In particular, it has been empirically shown [42], [46] that most program bugs are concentrated around the class boundaries (i.e., the extreme values).

In our running example, there are two input variables: year and month; and their corresponding equivalence classes are the following ones:

$$\begin{cases} \text{year} \begin{cases} \text{valid classes} \begin{cases} [1600, 3000] \\ \text{regular year} \\ \text{leap year} \end{cases} \\ \text{invalid classes} \begin{cases} < 1600 \\ > 3000 \end{cases} \end{cases} \\ \text{month} \begin{cases} \text{valid classes} \begin{cases} \text{months with 30 days} \\ \text{months with 31 days} \\ \text{February (28 or 29 days)} \end{cases} \\ \text{invalid classes} \begin{cases} < 1 \\ > 12 \end{cases} \end{cases} \end{cases}$$

To sum up, the instructor should define the equivalence classes for the program under consideration, and from them, derive the input values, which are the test cases prime material. Unfortunately, an exponential-growth problem arises again: a test suite including every possible combination of n variables, each one with $v_1, v_2, \ldots, v_n$ possible values, has size $v_1 \cdot v_2 \cdot \ldots \cdot v_n$. The calendar program only has two input variables, but more complicated programs may require a larger number of variables, ending up with a huge test suite. We manage this problem adopting the *combinatorial testing* approach.

Combinatorial testing is based on an interesting empirical fact [11], [47]–[50]: "most program failures appear to be caused by interactions of only a few variables, and hence tests that cover all such few-variable interactions can be very effective". From the input values the instructor provides, our framework generates a *pairwise* coverage, which includes every possible combination of values for each pair of variables. It is worth noting that a variety of experiments have shown that pairwise testing detects approximately 90% of the program failures [11], [51], reducing the test suite size considerably. If all n variables had v possible values, the test suite size for all possible combinations would be v^n, whereas for the pairwise combinations would be $v^2 \cdot \log n$ [51].

Our framework is currently implemented as an extension of the Code::Blocks open-source IDE [52], and it is freely available at:

http://www.issi.uned.es/fp/archive/cmasmenos.exe

IV. EMPIRICAL VALIDATION: THE 8-YEARS UNED EXPERIENCE

This section reports the application of our framework into a semestral subject of the Bachelor's Degree in Computer Science and Engineering at UNED. The subject fundamentally implements the *CS 115 Introduction to Computer Programming* course defined in the CS2013 [53] curricula recommendations given by the IEEE Computer Society and the Association for Computing Machinery.

The degree follows the European Bologna Declaration [54], which promotes the competency-based educational paradigm. The adoption of this paradigm supposed a great challenge: approximately 2,000 students had to be continuously evaluated by a teaching team composed of only three professors. To overcome this problem, we designed four voluntary programming assignments of increasing complexity: the first three assignments were automatically corrected with the framework described in Section III, and the last one by the teaching team. Whereas the results of the first assignments were Boolean (i.e., students' programs are functionally correct or not), the human corrected exercise received a numerical grade, which ranged from zero to ten and reflected the fulfillment of a variety of functional and non-functional quality attributes. Finally, students were allowed to carry out an assignment only if they had successfully passed the previous ones.

Students have a single submission date for each of the practices presented. In the case of AA practices, they receive feedback about whether their program works or not, and, if not, in what case it has failed. Completing these assignments means that they have been delivered on time and work correctly. In the case of the practice corrected by the teaching team, students receive the numeric grade and they have the possibility of carrying out a personal review to understand all their failures before the final exam.

A. EXPERIMENTAL SETUP

Our study tries to answer the following Research Questions (RQs), which are essential to judge our methodology:

- RQ1. Do programming assignments in general, and those automatically corrected in particular, have any influence on students' drop out?
- RQ2. Does our AA system properly prepares students to cope skillfully with complex assignments corrected by humans?
- RQ3. Does the assignments' completion influence the final exam grade?

Due to ethical restrictions, our empirical study did not follow an experimental design that supports concluding causal conclusions. Students instead of being randomly assigned to treatment and control groups, could freely decide to perform the practices.

The R language was used for data management and analysis [55]. In particular, the built-in `t.test` function and the following packages were used:

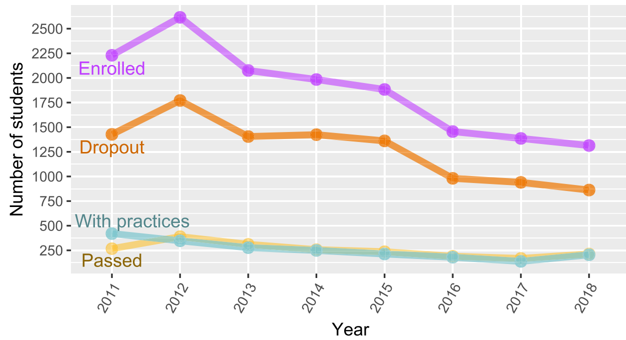

FIGURE 2. Number of students per year.

- `ggplot2` [56] for getting plots.
- `lsr` [57] for computing Cohen's d effect size.
- `car` [58] to perform the Levene's test for variance homogeneity.

B. RESULTS AND DISCUSSION

The results of our empirical study are available at:

https://github.com/rheradio/IntProgResults

Figure 2 shows the number of students enrolled in the course (in lilac color) from 2011 to 2018, distinguishing also the number of dropouts (in orange), the students who carried out the practices (in green) and, finally, the students who passed the final exam (in yellow).

1) RQ1. DO PROGRAMMING ASSIGNMENTS IN GENERAL, AND THOSE AUTOMATICALLY CORRECTED IN PARTICULAR, HAVE ANY INFLUENCE ON STUDENTS' DROP OUT?

Table 1 summarizes the descriptive statistics concerning our students' dropout. It confirms a distance learning critical problem: a remarkably high dropout rate, much higher than in traditional face-to-face education. Some studies indicate that dropout rates in e-learning are between 10% and 20% higher than in traditional education [59]. Other studies report even higher rates. For example, Simpson [12] reported a 78% dropout rate at the British Open University. Our rate (68.03%) is consistent with this last paper.

TABLE 1. Descriptive statistics for students' dropout.

Mean	Std. dev.	Median	Median abs. dev.	Min	Max
68.03	2.8	67.7	1.8	63.99	72.28

Figure 3 represents the number of students that completed and did not complete the assignments, distinguishing if they dropped out. The figure shows that (i) none of the students that finished the voluntary practices dropped out, and (ii) most of the students that did not complete the assignments dropped out.

TABLE 2. Inference statistics that confirm, every year, a difference between the students' final grades depending on whether they have completed the voluntary assignments.

Year	Practice?	Descriptive statistics: Sample size	Mean	Std. dev.	Median	Median abs. dev.	Min	Max	Levene's test p-value	t-test: p-vale	95% Conf. interval	Cohen's d
2011	No	385	1.57	1.56	1.2	0.89	0.3	8	<2.2e-16	<2.2e-16	[2.52, ∞)	1.19
	Yes	418	4.36	2.88	5	3.97	0.3	9.6				
2012	No	497	2.76	2.32	1.8	0.89	0.3	9	5.21e-05	<2.2e-16	[3.62, ∞)	1.60
	Yes	347	6.66	2.58	7.5	2.22	0.6	10				
2013	No	394	2.55	2.22	1.8	0.89	0.3	9	0.02	<2.2e-16	[3.75, ∞)	1.75
	Yes	277	6.6	2.44	7.1	1.63	0.9	10				
2014	No	313	2.52	2.41	1.5	0.89	0.3	9.3	0.13e-2	<2.2e-16	[2.90, ∞)	1.29
	Yes	247	5.78	2.65	6.1	2.82	0.3	10				
2015	No	310	2.6	2.11	1.8	0.89	0.3	9.1	0.03	<2.2e-16	[3.49, ∞)	1.76
	Yes	212	6.41	2.27	7	2.08	0.9	10				
2016	No	295	2.37	1.97	1.8	0.89	0.3	8.1	4.71e-08	<2.2e-16	[2.98, ∞)	1.50
	Yes	180	5.72	2.62	6.2	2.67	0.6	10				
2017	No	308	2.54	2.32	1.5	0.89	0.3	9	0.8e-2	<2.2e-16	[3.42, ∞)	1.62
	Yes	138	6.37	2.45	6.85	2.67	0.6	10				
2018	No	247	2.62	1.95	1.8	0.89	0.3	9	8.91e-05	<2.2e-16	[3.21, ∞)	1.67
	Yes	205	6.16	2.31	6	2.08	0.6	10				

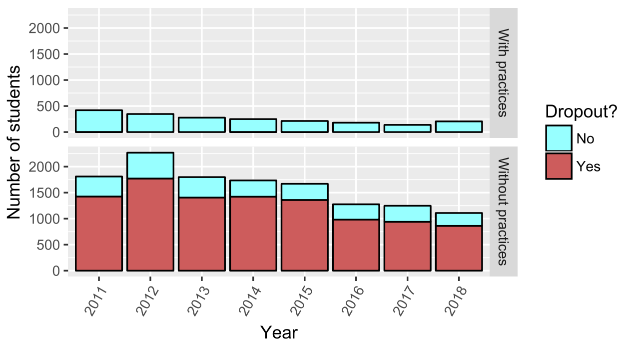

FIGURE 3. Amount of students that completed the assignments versus those that dropped out.

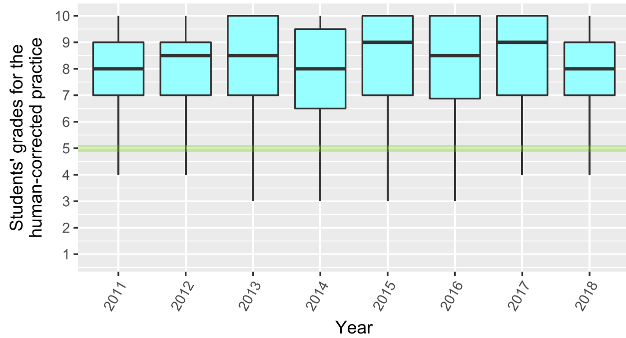

FIGURE 4. Grades of the practice corrected by the teaching team.

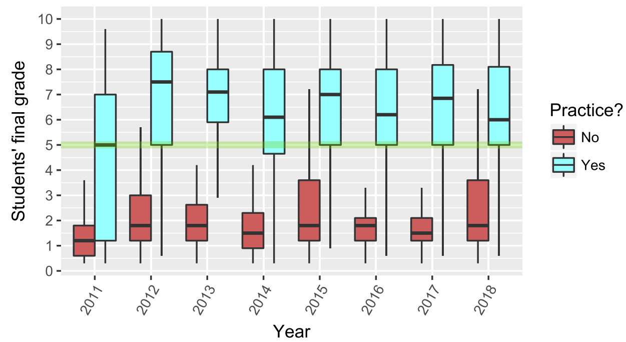

FIGURE 5. Students' final grades depending on whether they have completed the voluntary assignments (blue) or not (red).

2) DOES OUR AA SYSTEM PROPERLY PREPARES STUDENTS TO COPE SKILLFULLY WITH COMPLEX ASSIGNMENTS CORRECTED BY HUMANS?

The box-plot in Figure 4 summarizes the students' grades for the assignment corrected by the teaching team. As completing the first three practices was a mandatory requirement to carry out this last assignment, all students in the figure were trained with our AA system. As the figure shows, the system seems to adequately prepare students, since most of them obtained high grades.

3) RQ3. DOES THE ASSIGNMENTS' COMPLETION INFLUENCE THE FINAL EXAM GRADE?

Figure 5 compares the final exam grades of the students that completed the practices to those that did not. Table 2 confirms the visual information with inference statistics. Each year, the exam grades of both students' groups are compared. To do so, a t-test is performed for every year. But first, a Levene's test for variance homogeneity is undertaken to check if the t-tests need any adjustment. As all Levene's tests fail (the p-value is less than 0.5), the Welch's correction is applied. It can be seen that, in every course, the difference between the groups is statistically significant (p-value less than 0.5) and the effect size is enormous (Cohens's d much greater than 0.8 [60]). To sum up, there is a strong correlation between completing the assignments and obtaining high grades in the final exam.

V. CONCLUSION

Current assessment systems for computer programming practices cannot entirely replace human judgment, as they are unable to assess a variety of non-functional quality attributes that students need to acquire (e.g., program legibility, modular conceptual cohesion, etc.). In our university, we were aware of these shortcomings, but also of the many advantages that these systems provide.

In this article, we have presented a new automatic assessment system that is built upon software testing techniques, and reported its successful application into a rather populated course supported by a reduced number of teachers.

REFERENCES

[1] D. P. Moreira, "From on-campus to online: A trajectory of innovation, internationalization and inclusion," *Int. Rev. Res. Open Distrib. Learn.*, vol. 17, no. 5, pp. 1–14, 2016.

[2] A. M. Collins and R. Halverson, *Rethinking Education in the Age of Technology: The Digital Revolution and Schooling in America*. New York, NY, USA: Teachers College Press, 2018.

[3] *The Open University Facts*. Accessed: Jul. 19, 2019. [Online]. Available: http://www.open.ac.uk/about/main/strategy-and-policies/facts-and-figures

[4] *Universidad Nacional de Educación a Distancia Statistics*. Accessed: Jul. 19, 2019. [Online]. Available: https://app.uned.es/evacal/genmat.aspx

[5] *Coursera Info*. Accessed: Jul. 19, 2019. [Online]. Available: https://about.coursera.org/press

[6] *edX Info*. Accessed: Jul. 19, 2019. [Online]. Available: https://www.edx.org/about-us

[7] *XuetangX*. Accessed: Jul. 19, 2019. [Online]. Available: http://www.xuetangx.com/global

[8] S. Grover and S. Basu, "Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic," in *Proc. ACM SIGCSE Tech. Symp. Comput. Sci. Educ.*, 2017, pp. 267–272.

[9] D. Krpan, S. Mladenović, and M. Rosić, "Undergraduate programming courses, students' perception and success," *Procedia-Social Behav. Sci.*, vol. 174, pp. 3868–3872, Feb. 2015.

[10] T. Soffer and R. Nachmias, "Effectiveness of learning in online academic courses compared with face-to-face courses in higher education," *J. Comput. Assist. Learn.*, vol. 34, no. 5, pp. 534–543, 2018.

[11] D. R. Kuhn, R. N. Kacker, and Y. Lei, *Introduction to Combinatorial Testing*. Boca Raton, FL, USA: CRC Press, 2016.

[12] O. Simpson, "22%—Can we do better?" *CWP Retention Literature Rev.*, vol. 47, 2010.

[13] V. C. Lee, Y.-T. Yu, C. M. Tang, T.-L. Wong, and C. K. Poon, "ViDA: A virtual debugging advisor for supporting learning in computer programming courses," *J. Comput. Assist. Learn.*, vol. 34, no. 3, pp. 243–258, 2018.

[14] A. Bey, P. Jermann, and P. Dillenbourg, "A comparison between two automatic assessment approaches for programming: An empirical study on MOOCs," *J. Educ. Technol. Soc.*, vol. 21, no. 2, pp. 259–272, 2018.

[15] J. Hollingsworth, "Automatic graders for programming classes," *Commun. ACM*, vol. 3, no. 10, pp. 528–529, 1960.

[16] G. Singh, S. Srikant, and V. Aggarwal, "Question independent grading using machine learning: The case of computer program grading," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 263–272.

[17] J. Moreno-León, G. Robles, and M. Román-González, "Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking," *Rev. Educ. Distancia*, vol. 15, no. 46, pp. 1–23, 2015.

[18] Y. Wang, H. Li, Y. Feng, Y. Jiang, and Y. Liu, "Assessment of programming language learning based on peer code review model: Implementation and experience report," *Comput. Educ.*, vol. 59, no. 2, pp. 412–422, 2012.

[19] N. A. Rashid, L. W. Lim, O. S. Eng, T. H. Ping, Z. Zainol, and O. Majid, "A framework of an automatic assessment system for learning programming," in *Advanced Computer and Communication Engineering Technology*. New York, NY, USA: Springer, 2016, pp. 967–977.

[20] D. Thiébaut, "Automatic evaluation of computer programs using Moodle's virtual programming lab (VPL) plug-in," *J. Comput. Sci. Colleges*, vol. 30, no. 6, pp. 145–151, 2015.

[21] H. Kitaya and U. Inoue, "An online automated scoring system for java programming assignments," *Int. J. Inf. Educ. Technol.*, vol. 6, no. 4, p. 275, 2016.

[22] 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 *Proc. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, 2017, pp. 92–97.

[23] B. C. Wünsche, Z. Chen, L. Shaw, T. Suselo, K.-C. Leung, D. Dimalen, W. van der Mark, A. Luxton-Reilly, and R. Lobb, "Automatic assessment of OpenGL computer graphics assignments," in *Proc. 23rd Annu. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, 2018, pp. 81–86.

[24] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Comput. Sci. Educ.*, vol. 15, no. 2, pp. 83–102, Jun. 2005.

[25] M. Striewe and M. Goedicke, "A review of static analysis approaches for programming exercises," in *Proc. Int. Comput. Assist. Assessment Conf.* New York, NY, USA: Springer, 2014, pp. 100–113.

[26] H. Keuning, J. Jeuring, and B. Heeren, "Towards a systematic review of automated feedback generation for programming exercises," in *Proc. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, 2016, pp. 41–46.

[27] D. M. Souza, K. R. Felizardo, and E. F. Barbosa, "A systematic literature review of assessment tools for programming assignments," in *Proc. IEEE 29th Int. Conf. Softw. Eng. Educ. Training (CSEET)*, Apr. 2016, pp. 147–156.

[28] C. Douce, D. Livingstone, and J. Orwell, "Automatic test-based assessment of programming: A review," *J. Educ. Resour. Comput.*, vol. 5, no. 3, p. 4, 2005.

[29] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proc. 10th Koli Calling Int. Conf. Computing Educ. Res.*, 2010, pp. 86–93.

[30] C. G. von Wangenheim, J. C. Hauck, M. F. Demetrio, R. Pelle, N. da Cruz Alves, H. Barbosa, and L. F. Azevedo, "CodeMaster–automatic assessment and grading of app inventor and snap! Programs," *Inform. Educ.*, vol. 17, no. 1, pp. 117–150, 2018.

[31] G. Polito and M. Temperini, "A gamified approach to automated assessment of programming assignments," in *Challenges and Solutions in Smart Learning*. Singapore: Springer, 2018, pp. 3–12.

[32] D. Fonte, D. D. Cruz, A. L. Gançarski, and P. R. Henriques, "A flexible dynamic system for automatic grading of programming exercises," in *Proc. 2nd Symp. Lang., Appl. Technol.*, in OpenAccess Series in Informatics (OASIcs), vol. 29. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

[33] D. Jackson and M. Usher, "Grading student programs using ASSYST," in *ACM SIGCSE Bull.*, vol. 29, no. 1, pp. 335–339, 1997.

[34] A. Hentschel, C. Körner, R. Plösch, S. Schiffer, and S. Storck, "Method for the computer-assisted analysis of software source code," U.S. Patent 9 274 924, Mar. 1, 2016.

[35] S. M. Arifi, I. N. Abdellah, A. Zahi, and R. Benabbou, "Automatic program assessment using static and dynamic analysis," in *Proc. 3rd World Conf. Complex Syst. (WCCS)*, Nov. 2015, pp. 1–6.

[36] S. Krusche and A. Seitz, "ArTEMiS: An automatic assessment management system for interactive learning," in *Proc. 49th ACM Tech. Symp. Comput. Sci. Educ.*, 2018, pp. 284–289.

[37] D. Insa and J. Silva, "Automatic assessment of java code," *Comput. Lang., Syst. Struct.*, vol. 53, pp. 59–72, Sep. 2018.

[38] F. Restrepo-Calle, J. J. Ramírez Echeverry, and F. A. González, "Continuous assessment in a computer programming course supported by a software tool," *Comput. Appl. Eng. Educ.*, vol. 27, no. 1, pp. 80–89, 2019.

[39] L. de Oliveira Brandão, Y. Bosse, and M. A. Gerosa, "Visual programming and automatic evaluation of exercises: An experience with a STEM course," in *Proc. IEEE Frontiers Edu. Conf. (FIE)*, Oct. 2016, pp. 1–9.

[40] D. M. D. Souza, S. Isotani, and E. F. Barbosa, "Teaching novice programmers using ProgTest," *Int. J. Knowl. Learn.*, vol. 10, no. 1, pp. 60–77, 2015.

[41] N. Wirth, "Program development by stepwise refinement," *Commun. ACM*, vol. 14, no. 4, pp. 221–227, 1971.

[42] G. J. Myers, *The Art of Software Testing*. Hoboken, NJ, USA: Wiley, 2011.

[43] C. Kästner, "Differential testing for variational analyses: Experience from developing kconfigreader," 2017, *arXiv:1706.09357*. [Online]. Available: https://arxiv.org/abs/1706.09357

[44] M. A. Gulzar, Y. Zhu, and X. Han, "Perception and practices of differential testing," in *Proc. 41st Int. Conf. Softw. Eng., Softw. Eng. Pract.*, May 2019, pp. 71–80.

[45] D. Lehmann and M. Pradel, "Feedback-directed differential testing of interactive debuggers," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2018, pp. 610–620.

[46] C. Kaner, S. Padmanabhan, and D. Hoffman, *The Domain Testing Workbook*. New York, NY, USA: Context Driven Press, 2013.

[47] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Trans. Softw. Eng.*, vol. 30, no. 6, pp. 418–421, Jun. 2004.

[48] K. Z. Bell and M. A. Vouk, "On effectiveness of pairwise methodology for testing network-centric software," in *Proc. Int. Conf. Inf. Commun. Technol.*, Cairo, Egypt, 2005, pp. 221–235.

[49] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: A survey," *Softw. Test., Verification Rel.*, vol. 15, no. 3, pp. 167–199, 2005.

[50] D. R. Kuhn and V. Okum, "Pseudo-exhaustive testing for software," in *Proc. 30th Annu. IEEE/NASA Softw. Eng. Workshop*, Columbia, MD, USA, Apr. 2006, pp. 153–158.

[51] R. Kuhn, R. N. Kacker, and Y. Lei, "Practical combinatorial testing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST SP 800-142, 2010.

[52] *Code Blocks: Open Source, Cross Platform, Free C, C++ and Fortran IDE*. Accessed: Jul. 25, 2019. [Online]. Available: http://www.codeblocks.org/

[53] "Computer science curricula. Curriculum guidelines for undergraduate degree programs in computer science," Joint Task Force Comput. Curricula Assoc. Comput. Mach., New York, NY, USA, Tech. Rep. CS2013, 2013.

[54] "Joint declaration of the European ministers of education, the Bologna declaration," Eur. Ministers Charge Higher Educ., Belgium, Tech. Rep., 1999. [Online]. Available: http://www.ehea.info/cid100210/ministerial-conference-bologna-1999.html

[55] *R: A Language and Environment for Statistical Computing*, R Found. Stat. Comput., Vienna, Austria, 2014. [Online]. Available: http://www.R-project.org/

[56] C. Ginestet, "ggplot2: Elegant graphics for data analysis," *J. Roy. Stat. Soc., A, (Statist. Soc.)*, vol. 174, no. 1, pp. 245–246, 2011.

[57] D. Navarro, *Learning Statistics With R: A Tutorial for Psychology Students and Other Beginners*. Australia: Bookdown, 2018.

[58] J. Fox and S. Weisberg, *An R Companion to Applied Regression*, 2nd ed. Newbury Park, CA, USA: Sage, 2010.

[59] W. Doherty, "An analysis of multiple factors affecting retention in Web-based community college courses," *Internet Higher Edu.*, vol. 9, no. 4, pp. 245–255, 2006.

[60] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Evanston, IL, USA: Routledge, 1988.

DANIEL GALAN received the M.Sc. degree in automation and robotics from the Polytechnic University of Madrid, in 2013, and the Ph.D. degree in computer engineering and automatic control from the Universidad Nacional de Educación a Distancia (UNED), in 2017, where he is currently with the Department of Computer Science and Automatic Control, Computer Engineering School. His current research interests include robotics, virtual and remote labs, and virtual reality.

RUBEN HERADIO received the M.Sc. degree in computer science from the Polytechnic University of Madrid, Spain, in 2000, and the Ph.D. degree in software engineering and computer systems from the Universidad Nacional de Educación a Distancia (UNED), in 2007, where he is currently an Associate Professor with the Software Engineering and Computer Systems Department, Computer Engineering School. His current research and teaching interests include software engineering, computational logic, and e-learning.

HECTOR VARGAS received the degree in electrical engineering from the De la Frontera University, Temuco, Chile, in 2001, and the Ph.D. degree in computer science from the Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain, in 2010. Since 2010, he has been with the Electrical Engineering School, Pontificia UniversidadCattolica de Valparaiso. His current research interests include simulation and control of dynamic systems, multiagent systems, and engineering education.

ISMAEL ABAD received the Ph.D. degree in software engineering and computer systems from the Universidad Nacional de Educación a Distancia (UNED), in 2016. He belongs to the Software Engineering and Computer Systems Research Group. This research group has been involved in software engineering, robotics, and RFID research projects, since 2004. He is currently an Associate Professor with UNED. His current research interest include ubiquitous computing with hybrid systems (vision and RFID), and new software architectures for the Industrial IoT.

JOSE A. CERRADA received the M.S. degree in industrial engineering and the Ph.D. degree from the Polytechnical University of Madrid, Spain, in 1979 and 1983, respectively. He is currently a Full Professor and has been the Head of the Systems and Software Engineering Department, Universidad Nacional de Educación a Distancia (UNED), since 2015. His current research interests include RFID technologies and software engineering. He is teaching in the area of software engineering, specifically in the domain of software process management and improvement. He has participated in more than 30 research projects (European and Spanish Public Administration).