British Journal of Educational Technology *Vol 0 No 0 2019* 1–18
doi:10.1111/bjet.12756

Fruitful student interactions and perceived learning improvement in DSLab: A dynamic assessment tool for distributed programming

Joan Manuel Marquès Puig, Thanasis Daradoumis , Laura Calvet Liñan and Marta Arguedas

Joan Manuel Marquès Puig holds a PhD in Computer Science from the Universitat Politècnica de Catalunya. He is an Associate Professor at Computer Sciences, Multimedia & Telecommunication Studies at Universitat Oberta de Catalunya a (UOC) since 1997. His research interests include the design of scalable and cooperative Internet services and applications, privacy and e-learning. Thanasis Daradoumis holds a PhD in Computer Science from the Polytechnic University of Catalonia. He combines his role as Associate Professor at the University of the Aegean with his collaboration at the UOC. His research interests are: Emotional Intelligence, Alternative (Holistic) Education, Learning Analytics, E-learning, Collaborative, Affective and Adaptive Systems, CSCL. Laura Calvet Liñan holds a PhD in Network and Information Technologies from the Open University of Catalonia (UOC). She is a lecturer at the Department of Computer Science, Multimedia and Telecommunication (EIMT) at the UOC. Her research is related to Applied Statistics, Metaheuristics, Simulation, and e-Learning. Marta Arguedas holds a PhD in Education and ICT (eLearning) from the UOC. She is collaborating professor and tutor for the Master in Education and ICT at the UOC. Her research interests are: Emotional Intelligence, Learning Analytics, E-learning, Collaborative, Affective and Adaptive Systems, Interaction Analysis, CSCL, CSCW, Knowledge Management. Address for correspondence: Thanasis Daradoumis, Department of Computer Science, Multimedia and Telecommunications, Open University of Catalonia, Rambla Poblenou 156, 08018 Barcelona, Spain; Department of Cultural Technology and Communication, University of Aegean, University Hill, 81100 Mytilini, Greece. Email: adaradoumis@uoc.edu

Abstract

This work presents an automated assessment tool for online distributed programming, called DSLab. It is a web-based environment that provides a transparent deployment and execution of assignments in remote computers, a transparent initialization and the possibility to add new logs by the students. DSLab has been evaluated in a real distributed learning environment by analysing students' own perceptions on their learning improvement and exploring whether students' interactions with the tool yielded a fruitful learning experience. Current research provides no evidence that the effects of such a tool have been investigated in the field of online distributed programming. Two types of analysis were performed: a quantitative analysis of data coming from students answering a questionnaire and an analysis of the log files of students' interactions with the tool. Our results showed that students perceived substantial learning improvement from the use of the automated assessment tool. Moreover, students produced fruitful interaction with the tool as soon as they achieved high familiarization and constant activity with it, which ultimately helped them improve their academic performance. Finally, the limitations of the current study and directions for future research are presented.

Practitioner Notes

What is already known about this topic

- Several automated assessment tools of programming assignments have been developed during many years.
- Assessing programming assignments still remains a labour-intensive task.
- Current research provides no evidence that the effects of such a tool have been investigated in the field of online distributed programming (DP).

What this paper adds

- Proposes a new automated assessment tool (AAT) for online DP, called DSLab.
- DSLab provides a transparent deployment and execution of assignments in remote computers, a transparent initialization and the possibility to add new logs by the students.
- Students perceive substantial learning improvement from the use of DSLab.
- Students produce fruitful interaction with the tool as soon as they achieve high familiarization and constant activity with it.
- DSLab allows students to assess their code in a consistent manner, get immediate feedback, strive for a better solution and better learning, by submitting their code multiple times and improving it through the tool feedback.

Implications for practice and/or policy

- Supports both students in resolving complex DP assignments and teachers in evaluating time-consuming and labour-intensive tasks.
- Provides an approach and methodology for analysing the effects of AATs of DP assignments on students' academic performance.
- Underlines the need for more rigorous experiments that can explore the effects of AATs of DP assignments on student learning in more depth.

Introduction

In Distributed Systems courses, both teaching and learning processes have to be assisted by adequate methods and tools that enable students to acquire the necessary distributed programming skills and thus produce robust and efficient code. Providing students the capability of submitting their assignment multiple times would improve the code and support iterative learning, especially if they use the feedback that a self-assessment tool provided in previous executions (Suleman, 2008).

Though several automated assessment tools of programming assignments have been developed during many years, assessing programming assignments still remains a labour-intensive task (Pettit, Homer, Holcomb, Simone, & Mengel, 2015). This is especially evident in distributed programming, where, as far as we know, tools for self-assessing distributed programs sin real long-term online educational settings are hardly found. Consequently, we developed DSLab, a web-based tool that students can use for self-assessing distributed applications they build using a real distributed deployment under realistic conditions. All the details about the DSLab web interface, features and a real example of a student's assignment are presented in Online Appendix A.

We argue that the benefits of online students employing this tool are manifold: (1) their solutions are assessed in a consistent manner and get immediate feedback, which enables them to know whether their code doesn't work adequately, (2) they strive for a better solution and, therefore,

better learning, by submitting their code multiple times and improving it through the tool feedback and (3) they are provided with a grading facility, which lets them know the mark they obtain at the end of the execution. Since their assignment consists of different phases, students can decide when to stop because they have reached the desired mark (an interesting side effect for online students that simultaneously study and work, which lets them manage their workload).

The rest of the paper is organized as follows: Section 2 presents related work on automated assessment tools of programming, emphasizing the lack of such tools in distributed programming; then it defines the purpose of our study and research questions. Section 3 describes our automated assessment tool. Section 4 presents the implementation and experimentation of our tool. The results of this experimental study are presented in Section 5. In Section 6, we discuss and analyse these results with regard to the research questions. Finally, conclusions and future work are presented.

Literature review

An early review of Ihantola, Ahoniemi, Karavirta, and Seppälä (2010) revealed that various automated assessment tools of programming assignments were developed for many years; however, most of them had a limited lifespan due to the short-term or limited nature of the overall system. Later, the need for effective assessment of programming skills of MOOC students has shifted interest to more practical, intuitive and automated online assessment of programming (Staubitz, Klement, Renz, Teusner, & Meinel, 2015).

Moreover, Git is an open-source version control system which students use to submit and check their codes continuously until the code meets the assignment requirements (Kelleher, 2014; Lawrance, Jung, & Wiseman, 2013; Loeliger & McCullough, 2012). However, Git is not a fully automated tool, since there are several tasks that teachers and students have to perform manually. An extension of Git (a platform, called ProgEdu), provided by Chen, Chen, Hsueh, Lee, and Li (2017), enables automatic assessment of assignments in programming courses; however, their tool has not been tested in real long-term educational settings.

Recent assessment tools utilize new mechanisms for assessing programming assignments, examining the type and nature of student errors and providing meaningful feedback, such as edit distance (Barker-Plummer, Dale, & Cox, 2012), control flow graph similarity measurement (Vujošević-Janičić, Nikolić, Tošić, & Kuncak, 2013), or Euclidian and Levenshtein word distance (Stajduhar & Mausa, 2015). However, these systems fail to detect "informal" errors, but most importantly they lack of a systematic way to cope effectively with the whole formative assessment process. Our work presents an initial effort to fill this gap by analysing the factors and features that can contribute toward a systematic development of an efficient and useful assessment tool in distributed programming.

Virtual programming lab systems and other programming assessment plugins have also been implemented over Course Management Systems (CMS), like Moodle, such as: the Virtual Programming Lab (VPL) developed by Rodríguez-del-Pino, Rubio-Royo, and Hernández-Figueroa (2012), a tool for automated evaluation of VHDL and Matlab programming assignments (Ramos, Trenas, Gutiérrez, & Romero, 2013), and an online compiler and plagiarism detection tool for assessing programming assignments (Kaya & Özel, 2014). However, these tools are still far from providing an environment for a systematic assessment of programming.

Automated assessment tools have also been implemented as part of bigger Intelligent Tutoring Systems (ITS). As such, ITSs are capable of: employing systematic assessment and grading methods (Baker & Rossi, 2013); allowing students to correct their errors in real time (Malmi *et al.*,

2004); offering students appropriate formative feedback to enhance their knowledge construction (Clark, 2012; Heffernan & Heffernan, 2014); and updating students' models and making them aware of their misconceptions, knowledge gaps and their own learning more accurately and in a timely manner (Grivokostopoulou, Perikos, & Hatzilygeroudis, 2017). Other tutoring systems apply similar data-driven approaches that use hints to incrementally generate a solution space for programming problems, supporting different computer science competencies, from deductive logic (Mostafavi & Barnes, 2017), Python Programming (Rivers & Koedinger, 2017) to functional programming (Gerdes, Heeren, Jeuring, & van Binsbergen, 2017), leading to model correct student programs.

Robinson and Carroll (2017) describe an open-source online learning platform that provides new methods for automated formative and formal summative assessment, allowing both standard compiler/interpreter feedback and customized contextual instructor guidance. The system can also keep track of students' actions and the resulting analysis can provide information about student participation and progress. However, a real appreciation of the system's utility and efficiency is lacking, since an in-depth evaluation of the perceptions of its users is needed.

Pettit *et al.* (2015) conducted an interesting survey into the real usefulness of automated assessment tools (AAT) in programming courses. They discovered that AATs are helpful in improving student learning, supporting instructors, and assuring assessment accuracy. However, they also found that students' opinions about the helpfulness of AATs are inconclusive.

Recently, some web tools have been created that include frameworks to run students' assignments in distributed computational resources in a transparent and less cumbersome manner, relieving also instructors or administrative staff of creating a remote login for each student and then training them to use the running environment. Maggi and Sisto (2007) present a framework to create web portals for running assignments on distributed computers automatically. Foley *et al.* (2017) present OnRamp, a web portal that allows students to have an easy hands-on exploration of parallel and distributed computing concepts. Sukhoroslov (2018) presents a more advanced design and implementation of a web-based environment on top of a general-purpose platform Everest (Sukhoroslov, Volkov, & Afanasiev, 2015) for executing students' assignments in parallel and distributed computing.

On the one hand, the design of our DSLab tool was based on our analysis of the literature review, more specifically, of the characteristics that were developed by related assessment tools for distributed programming. On the other hand, through experimentation with our tool, we identified some more important features that might facilitate the use of the tool by students and improve the learning process through feedback, like transparent initialization, and the possibility to add new logs by the students.

Table 1 outlines the strengths of our DSLab tool by comparing DSLab features with those of the existing software.

Based on DSLab functioning described in Online Appendix A, Figures 13 and 14 present the web-based environment that DSLab offers to students to execute their Java-based programming assignments and view the results generated by the correction process (features 1, 2 and 4). Moreover, Figure 16 presents the library that transparently initializes the code to be executed and coordinates its execution (feature 3), whereas Figures 15 and 17 show an extract of the logs generated during an execution and how students can add new logs to be collected during the execution respectively (feature 5). Features 3 and 5 are two distinctive attributes of DSLab that differentiate it from the existing systems presented in Table 1.

Table 1: Comparison of DSLab features with existing software

Feature software	*1. Web-based environment for transparent deployment and execution of assignments in remote computers*	*2. Supported programming languages*	*3. Library that transparently initializes the code to be executed and coordinates its execution*	*4. Access to the files or data generated by the correction process*	*5. Students can add new logs to be collected during the execution and access them from the web interface*
Maggi and Sisto (2007)	YES	Java	NO	YES	NO
OnRamp Foley *et al*. (2017)	YES	any	NO	YES	NO
Sukhoroslov (2018)	YES	any	NO	YES	NO
DSLab	YES	Java	YES	YES	YES

Eventually, despite the large number of automated assessment tools and published papers, the above survey showed that only a small percentage of the current research work includes formal results about the effects of tool use. Hence, the authors underline the need for more rigorous experiments that can explore the effects of AATs on student learning in more depth.

Purpose of the study

To fill this gap, this study examines the effects of our automated assessment tool on students' academic performance in a real distributed programming environment based on the analysis of students' perceptions of their learning improvement and the analysis of their interactions with the tool. The former is explored by quantitative analysis of data coming from students answering a questionnaire. The latter is achieved through an analysis of the log files of students' interactions with the tool. In particular, the study seeks to answer the following research questions:

- RQ1 – Did the students perceive learning improvement from the use of the DSLab automated assessment tool?
- RQ2 – Does a *fruitful interaction* occur between students and the tool which helps them improve their academic performance?

In this work, a "fruitful interaction" can be related to two important aspects:

- Students' success: in terms of successful executions of their assignments; the number of executions and their failures as these executions evolved during the learning activity; and, the number of times a student tried to carry out an execution in each particular phase.
- Students' learning: in terms of the effect of studying the execution logs on passing or failing the assignments; the effect of starting using the tool as early as possible on improving their learning and marks; and, the effect of working in group on achieving better interaction and learning results than those who did the assignments alone.

Research design and testing

Participants and experimental procedure

Our case study was performed in an undergraduate Distributed Systems online course at our university. Students had to implement a distributed algorithm, upload and execute their code to

DSLab and assess its correctness. The multiple submissions capability of DSLab—which allows students to improve their code using the feedback from previous executions—supports iterative learning (Suleman, 2008).

The assignment consisted of several implementation phases that were assessed using DSLab (phases: 2, 3 and 4.1). Each phase is an extension of the previous one, has an increasing level of difficulty and is assessed separately as it uses different configuration and execution conditions.

More specifically, the aim of the assignment was to implement and evaluate the weak-consistency protocol Time Stamped Anti-Entropy (TSAE) (Golding, 1992). In this assignment, TSAE was used to maintain the consistency between replicas of an application that stores cooking recipes in a set of servers. TSAE uses three data structures: a Vector clock, a Matrix clock and a Log of operations. The assignment has then been divided in three different phases:

- *Phase 2*: Implementation of a reduced version of the TSAE protocol to support only the *Add* operation. This is done by implementing the consistency sessions and the Vector clock as well as the Log data structure. In this phase, the Log data structure was never purged.
- *Phase 3*: Extension of Phase 2 to purge the Log data structure with unsynchronized clocks. This is done by extending the consistency sessions to include the purging functionality, implementing the Matrix clock and extending the Log data structure to allow purging.
- *Phase 4.1*: Implementation of the *Remove* recipe operation.

The course was carried out fully online for seven weeks. A total of 110 adult students participated, divided into three classes. All students performed the same assignment. Among the students, 13 were female (11.81%) and 97 were male (88.18%).

The assignment could be implemented individually or in group of two members, applying the same task and assessment criteria. In each assignment phase, students could submit their code as many times as they wished. For each submission, they received DSLab assessment and feedback, and thus could look at the execution logs to check for possible errors.

At the end of the course, students answered a customized questionnaire that was specifically designed to respond to RQ1, whereas log data were collected and analysed with the aim to respond to RQ2.

Data collection

The 110 students who participated in the study were asked to answer the questionnaire anonymously at the end of the learning activity on a voluntary basis. The questionnaire was finally answered by 54 students, that is, about 49% of the participants. This sample can be considered random since any participant could have decided to answer it independently whether he/she has completed all the assignments or not. The questionnaire included nine items divided into two categories (as shown in Table 2): six items related to the tool use (MU) and three items related to the tool feedback (MF). We used a five-point Likert-type scale ranging from 1 (Strongly disagree) to 5 (Strongly agree), requiring a quantitative answer.

Regarding the statistics employed in the questionnaire data analysis, we used descriptive statistics, calculating relative frequencies (%), as well as bivariate correlation and variance analysis to find relationships between the variables under study.

Additionally, tool log files, which automatically record interactions between tool and students, provided a rich data set for analysing the way students carried out a fruitful interaction with the tool.

Table 2: Questionnaire of nine (9) question items related to students' Learning iMprovement (RQ1)

Use of the DSLab automated assessment tool	
MU1	The fact of using the DSLab automated assessment tool has contributed to that you have learned to write better distributed algorithm program codes?
MU2	Have you learned the required concepts better and more easily by using the DSLab tool (which would not have happened if you had not used it)?
MU3	Do you think you have got a better mark by using the tool (which you would not have achieved if you had not used it)?
MU4	The fact that DSLab tool allowed you to perform multiple submissions of your assignment has caused you to work carelessly during the initial submissions?
MU5	The fact that the tool allowed you to make multiple submissions has prompted you to follow a standard coding pattern, re-submission and correction according to the response (the result and the log traces) you received from the tool? (which prevented you thinking of other actions that could be different from the ones suggested by the interpretation of the logs).
MU6	When you have found yourself in difficult situations doing the assignment, do you think that the use of the tool has helped you not to abandon the assignment and continue to learn and improve?
Feedback provided by the DSLab tool	
MF1	The logs associated with each execution that the tool gives you have served to improve your performance?
MF2	The logs have been difficult to understand?
MF3	Would you have preferred the logs to be much more descriptive with a log level aimed at novice programmers?

Results

The results with regard to RQ1

We analyse students' perceptions about their learning improvement in relation to two parameters: (1) the use of the DSLab tool by evaluating items MU1-MU6 and (2) the feedback provided by the DSLab tool by analysing items MF1-MF3 (see the corresponding descriptive statistic measures in Table 3 and Figure 1).As concerns both MU1 and MU2, 45% of the students (strongly) agree that the tool has helped them learn better and more easily the concepts required to program the distributed algorithms effectively. It is also important to note that nearly 30% of the students were rather neutral when asked whether they perceived DSLab to improve their programming knowledge and skills.

With respect to MU3, 53% of the students (strongly) agree that the tool has helped them get a better mark using the tool. Again, DSLab appeared to have little impact on the improvement of the mark of almost 25% of the students.

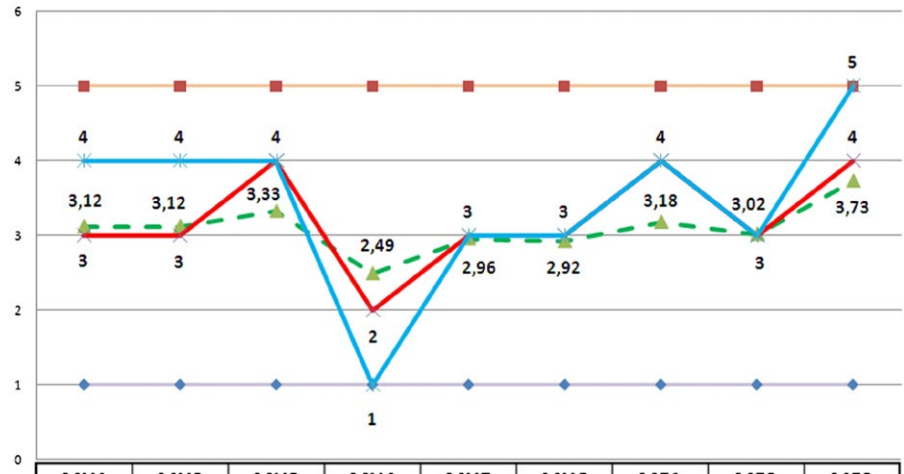
In relation to MU4, more than 61% of the students did not perceive the multiple submission feature of the tool as a factor to neglect their work. However, nearly 37% of the students had this negative perception.

As regards MU5, students' perceptions were divided in almost three equal parts; that is, only around 33% believed that the continuous use of the tool caused them to adopt standard coding patterns, suggested by the tool logs, which tend to be more logical and thus restricted their own coding creativity. The rest of the students were either neutral or negative to that perception.

Table 3: Frequency data concerning learning iMprovement based on DSLab Use & Feedback

	MU1		*MU2*		*MU3*		*MU4*		*MU5*		*MU6*		*MF1*		*MF2*		*MF3*	
	Fr	*%*	*Fr*	*%*	*Fr*	*%*	*Fr*	*%*	*Fr*	*%*	*Fr*	*%*	*Fr*	*%*	*Fr*	*%*	*Fr*	*%*
1	9	16,3	9	16,3	8	14,3	19	36,7	8	14,3	6	10,2	9	16,3	3	4,1	7	12,2
2	6	10,2	6	10,2	5	8,2	13	24,5	10	18,4	10	18,4	7	12,2	17	32,7	4	6,1
3	15	28,6	15	28,6	13	24,5	2	2,0	18	34,7	24	46,9	10	18,4	18	34,7	8	14,3
4	18	34,7	18	34,7	19	36,7	14	26,5	12	22,4	10	18,4	22	42,9	8	14,3	16	30,6
5	6	10,2	6	10,2	9	16,3	6	10,2	6	10,2	4	6,1	6	10,2	8	14,3	19	36,7

Note. The grey shaded values in Table are used to highlight the most significant values obtained.



	MU1	MU2	MU3	MU4	MU5	MU6	MF1	MF2	MF3
Min	1	1	1	1	1	1	1	1	1
Max	5	5	5	5	5	5	5	5	5
Mean	3,12	3,12	3,33	2,49	2,96	2,92	3,18	3,02	3,73
Median	3	3	4	2	3	3	4	3	4
Mode	4	4	4	1	3	3	4	3	5

Figure 1: The descriptive statistic measures concerning learning iMprovement based on DSLab Use & Feedback

Finally, in MU6, only 25% of the students had a positive perception that the tool was a good reason for not quitting but continuing their efforts despite the difficulties. More than 28% had just the opposite perception, whereas 47% expressed neutrality.

In conclusion, MU1, MU2 and MU3 show that the use of DSLab automated assessment tool in principle contributes to the improvement of the learning and academic performance of a remarkable number of students; however, a rather small number of students praised the tool's ability to help overcome very difficult situations (MU6). In addition, a large number of students considered that the multiple submission feature of the tool neither caused them to neglect their work nor to restrict their own coding creativity (MU4 and MU5); however, the rest of the students (about 33–37%) who had an opposite perception cannot be considered insignificant either.

With respect to the items that evaluated the feedback provided by the tool, their analysis presents the results shown below.

As concerns MF1, 53% of the students considered that the feedback offered by the tool in each execution through its logs has definitely helped them improve their performance. However, there was a number of students (a bit less than 30%) that had a negative perception of the feedback utility, a fact that is worthy of attention.

The latter is also confirmed regarding MF2. A bit less than 30% of students (strongly) agreed that the logs have been difficult to understand. However, about 37% of students had no problem of interpreting the logs, whereas the rest of students (about 35%) expressed neutrality in this matter.

Finally, as regards MF3, a large number of students (more than 67%) prefer that the type of feedback logs offered by the tool be much more descriptive at a level oriented to novice programmers. Only about 18% of students had no problem with the current log format.

In conclusion, the feedback offered by the tool was highly appreciated by the students to improve their algorithms and re-run their code until they achieve optimal performance of their algorithms. However, a notable number of students (around 30%) had to make a big effort in order to understand the logs and thus achieve some improvement in their performance. In this sense, a lot of students expressed their preference for more descriptive and explanatory logs, an issue that should be addressed in future versions of the tool.

Moreover, in order to provide a more comprehensive response regarding *the students' perceptions on learning improvement from the use of the tool*, we analysed the Pearson correlations between the two dependent variables: the *use* of the tool (MU) and the tool *feedback* (MF) with respect to the independent variable learning *improvement* (M), so that to identify the strong positive or negative linear relationships that exist among these variables (see Figure 2). We analyse these results in the Discussion section.

The results with regard to RQ2: Log data analysis of students' behaviour

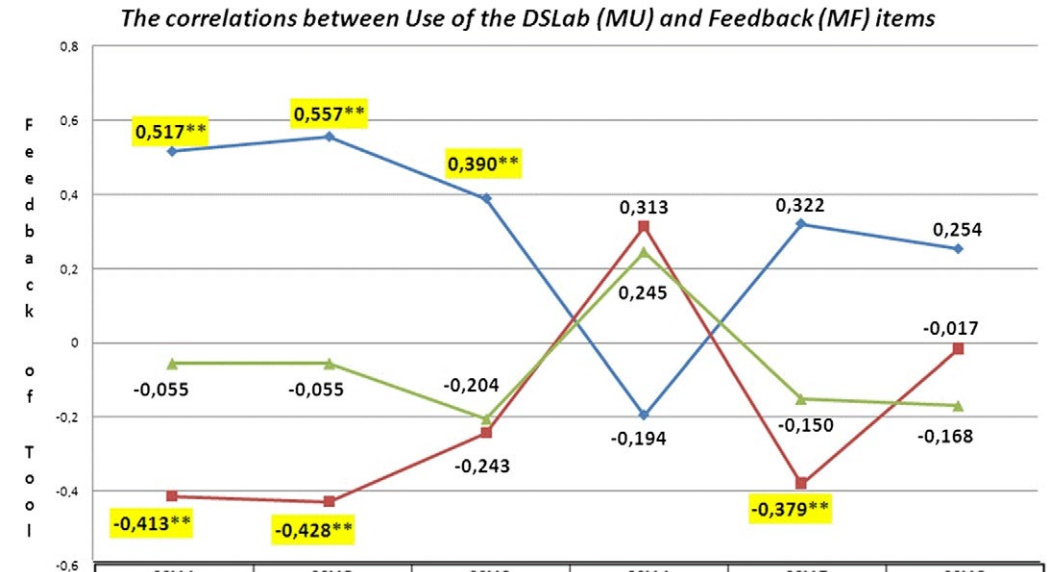
Here we describe the use of DSLab tool by students throughout the entire learning activity.

A total of 1340 executions were carried out by 82 participants, where participants were either individuals or groups of two students (54 individuals and 28 groups). Thus, there were 16.34 executions per participant on average. The percentage of executions by phase (2, 3 or 4.1) is relatively similar (29.85%, 36.19%, and 33.96% respectively); being the highest that of phase 3. As shown in Figure 3, the proportion of successful executions is higher in phase 2, whereas it is relatively similar in the others.Figure 4 depicts the number of executions during the learning activity by phase. A few participants start to execute soon but the number of executions gets much higher during the last month of the course. As expected, at the beginning participants focus on phase 2, but at the end there are executions of all phases. Figure 5 shows this evolution highlighting the proportion of failures.Eighty-two participants run executions for phase 2, but only 80% and 56% of them continue to the phase 3 and phase 4.1 respectively. Boxplots in Figure 6 show the distribution of the number of executions per participant for each phase (considering those participants that may execute 0 times the code for a given phase) and the total. The distributions are highly asymmetric. Interestingly, the mean value associated with the last phase is the lowest (many participants don't even try). According to the boxplot at the left, there are five participants that run their codes around 50 times or more. Moreover, 50% of the participants run them more than 10 times.Taking into account that only one successful execution was required to pass each phase, the percentage of participants that passed compared to those who tried was: 91%, 75%, and 70% respectively.

Table 4 shows the mean number of executions and the standard deviation (in parentheses) by phase and result. It can be concluded that those who passed the assignments had made more executions on average, but they also had much more variability (ie, most students did a relatively low number of executions while others did many more).

Regarding the execution logs, a total of 2326 logs were read by 48 participants (49.5 on average). The proportion of logs read belonging to each phase was: 41%, 46% and 12% respectively. Most of the logs read fall into the following categories: info (28%), trace (26%) and debug (22%).

The effect of reading the logs on passing or failing the assignments is inconclusive. For the first phase, the number of participants not passing is too low to make comparisons (7 participants). For the second phase, 81% of the participants who read the logs passed, whereas 72% passed the assignment without readings the logs. In the third phase, only 14 participants read the logs and 64% passed the assignment, whereas 72% passed the assignment without readings the logs. In



	MU1	MU2	MU3	MU4	MU5	MU6
MF1	0,517**	0,557**	0,390**	-0,194	0,322	0,254
MF2	-0,413**	-0,428**	-0,243	0,313	-0,379**	-0,017
MF3	-0,055	-0,055	-0,204	0,245	-0,150	-0,168

Figure 2: The correlations between Use of the DSLab (MU) and Feedback (MF) items

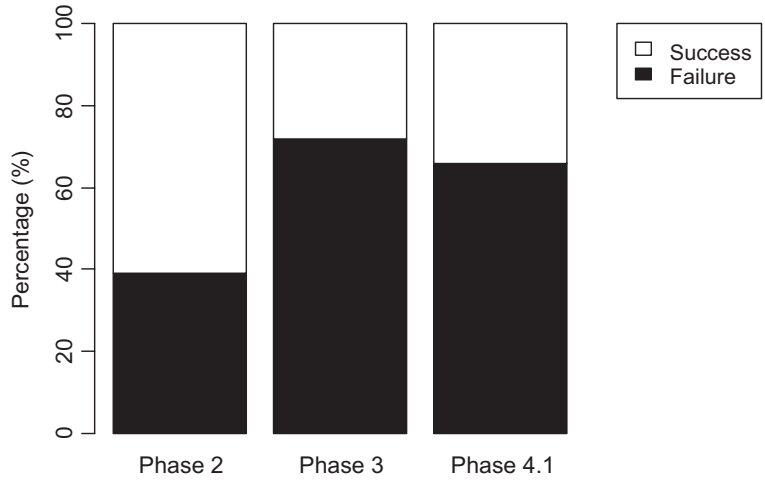


Figure 3: Percentage of successful executions for each assignment

our opinion, this "incoherence" is due to two opposing trends: (1) students that read more logs may have more time to do the assignments and are more motivated; and (2) students that read more logs do that because they may have problems solving the assignments as well as fewer clues about how to solve the mistakes.

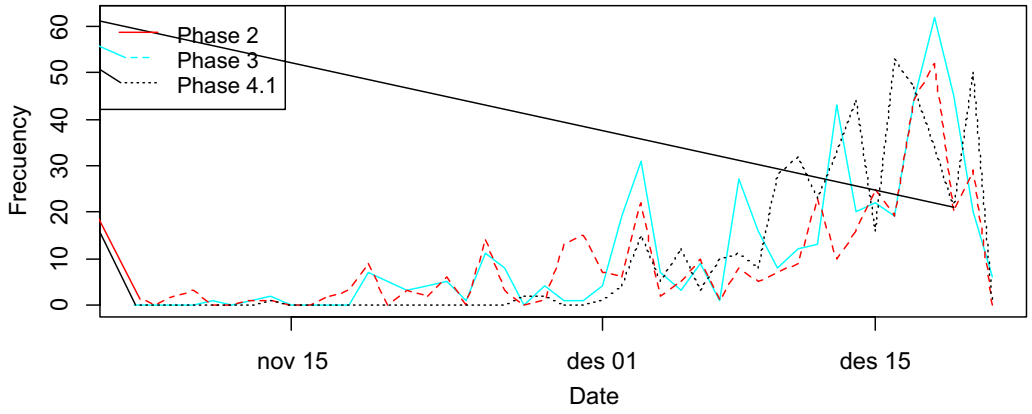


Figure 4: Evolution of the number of executions during the learning activity by phase

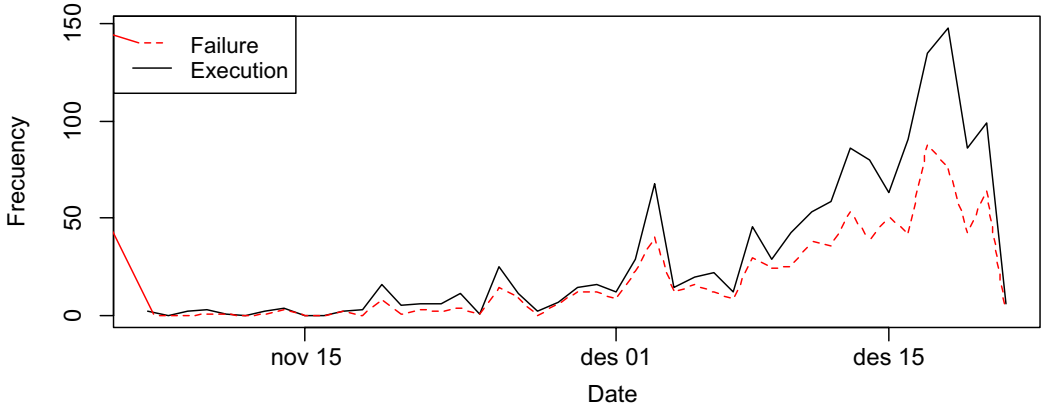


Figure 5: Evolution of the number of executions during the learning activity by result

We also analyse which is the effect of starting using the tool at the very beginning of the learning activity on improving student marks. First, we compute the median of the day when each participant starts using it. Then, the participants are split according to whether they started before or after the median—participants B (before) or A (after) respectively. Figure 7 shows the number of participants per result, considering the phase and participants. As expected, those who started before the median (participants B) had a better result.Finally, we compare those who do the assignment individually (I) versus those who do it in group (G). The median value of groups' starting date is December 6, whereas that of those doing the assignments alone is December 15, ie, one week and a half later. In general, groups achieve better results (Figure 8).

Discussion

In the previous section, the presentation of questionnaire results based on descriptive statistics as well as the analysis of log data provided us insights about the perceptions of students on their

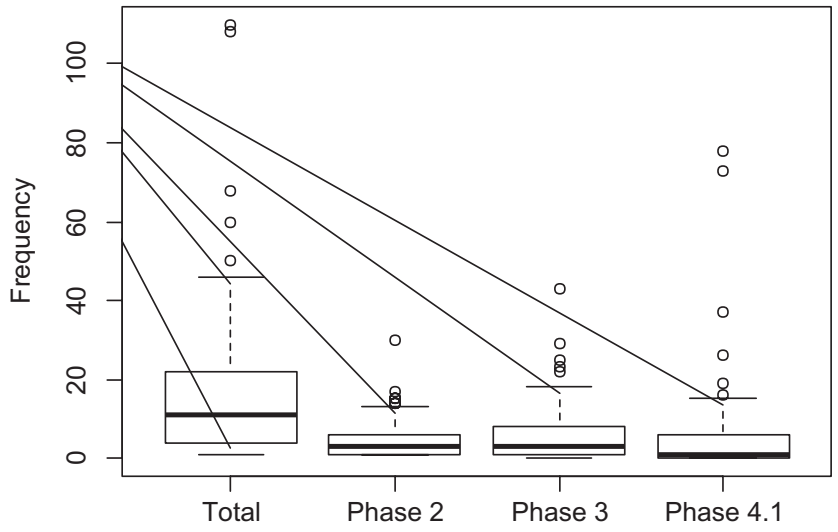


Figure 6: Boxplots of the number of executions by phase

Table 4: Mean number of executions made by phase and result (standard deviation in parentheses)

	Phase 2	*Phase 3*	*Phase 4.1*
Fail	3.29 (1.80)	4.06 (4.33)	4.86 (5.17)
Pass	5.03 (5.14)	8.40 (8.57)	12.09 (18.46)

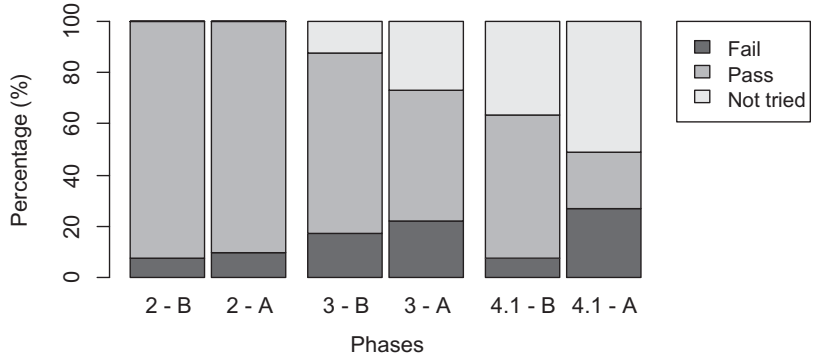


Figure 7: Results by phase and starting date (before or after the median)

learning improvement (RQ1) and whether students' interaction with the tool resulted in a fruitful learning experience (RQ2). Based on these results, in this section we attempt to take a closer look at the two research questions within the context of the existing literature.

Analysis for RQ1

The results of the Pearson correlations presented in Figure 2 (in the Results section) show that there is a strong positive relationship between variable MF1 and variables MU1, MU2 and MU3.

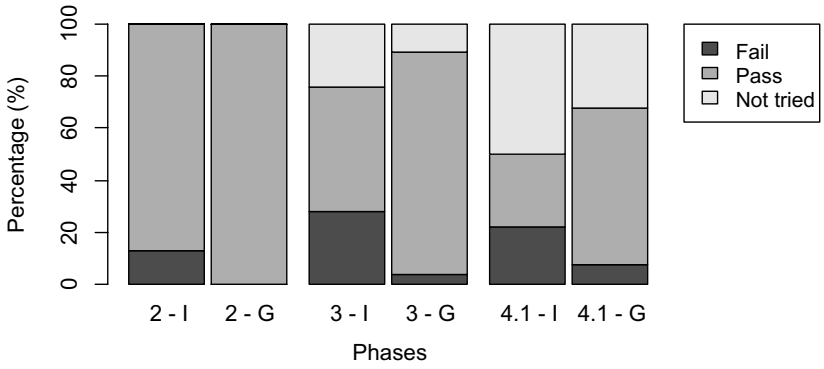


Figure 8: Results by phase differentiating between individuals (I) and groups (G)

This means that, for most students (<65%), the feedback they receive with each execution has served to improve their learning. This finding is consistent with previous research that revealed a positive impact on student learning with the introduction of an automated assessment tool into a programming course (Chen *et al.*, 2017; Kelleher, 2014; Pettit *et al.*, 2015). Unique to this study was the differentiation of our analysis between three fundamental aspects that influence learning improvement: write better distributed algorithm program codes, learn the required concepts better and more easily, and get a better mark. Tool feedback helped improve students' performance in all three aspects. Unlike previous research (Malmi *et al.*, 2004; Rodríguez-del-Pino *et al.*, 2012), where it was observed that "resubmissions can both cause students to work carelessly during early submissions and also create a pattern of coding, resubmitting, and fixing in response to feedback without putting any thought into actions outside of what the tool suggests," our study showed that: on the one hand, more than 61% of the students did not perceive that the multiple submission feature of the tool caused them to neglect their work; on the other hand, there is some kind of relationship between variables MF1 and MU5 which indicates that feedback had only a relative influence on prompting students to follow a standard coding pattern. While more research is necessary to confirm and better understand this relationship, it may not be too early to suggest that tool feedback has in general not hindered students' own creativity as a counterpoint for their learning improvement.

In the previous section, our analysis showed that almost 30% of students agreed that the logs have been difficult to understand, whereas about 35% of students expressed neutrality. In line with previous studies (Becker *et al*., 2016; Denny, Luxton-Reilly, & Carpenter, 2014), our analysis shows that there is a strong negative relationship between variable MF2 and variables MU1, MU2 and MU5. This indicates that several students had difficulty to understand the current format of feedback logs, which in turn has negatively influenced both learning the theoretical concepts and writing better code, since they were unable to follow the correction suggested by the result and the log traces of the feedback. This confirms our previous hypothesis that students would have preferred more extensive and specific feedback related to the detected errors and a clearer guidance to error correction (MF3). In fact, the absence of relationships between MF3 and all MU variables indicates that the current feedback content has to be significantly improved in order to cause major benefits to learning improvement.

Analysis for RQ2

With regard to the second question (*Does a fruitful interaction occur between students and the tool which helps them improve their academic performance?*), at a first level of log data analysis, it is

interesting to highlight that most executions were performed during the last month of the semester (Figure 4), when students focus more their efforts to cope with and pass the assignment, or to improve their academic performance. This fact suggests that student do perceive the usefulness of the tool and make the most of it during the last month of their learning activity.

Moreover, while students perform the assignments progressively, starting from the easiest one (Phase 2), the proportion of successful executions is constant through the semester (Figure 5). This fact may indicate that a relative progress is achieved. However, the dropout rates are not insignificant: 20% of the students who execute code for the first assignment do not even try the second one; similarly, 24% of the latter do not try the last one. This finding is consistent with the study of Rubio-Sanchez, Kinnunen, Pareja-Flores, and Velázquez-Iturbide (2014) that showed no significant difference in the dropout rate between a course that used an automated assessment tool and a course that did not. Our analysis suggests that dropout occurs because each assignment is an extension of the previous one, the level of difficulty is progressively increased, and some participants start to make executions relatively late, so they end up without enough time.

As in previous research (Robinson & Carroll, 2017; Stajduhar & Mausa, 2015; Staubitz *et al.*, 2015), those who pass the assignments have carried out more executions on average (though the dispersion was higher too). Thus, the more students use the tool the more possibilities they have to complete their assignment successfully. In our study, regarding the built-in event log reports of the tool, more than half of the participants have read at least one log report. This means that the majority of students found the logs useful in their effort to recover from errors and improve their code, so they learnt something from log use. However, there is no a clear relationship between checking the logs and passing the assignments. Log data display and exploitation is an issue that surely needs more improvement, so more work has to be done in order to improve the presentation of log results.

It can be also noted that students who started using the tool at the very beginning of the learning activity, they first get used to the tool, then they get more engaged in using it and finally they have more chances to complete their assignment successfully and pass the course (Figure 7).

At a second level of log data analysis, it has been found that the number of executions per participant (16.34 on average) is high. This means that students found the tool engaging or they were curious to try it at least.

Phase 3 is the phase where the most failed executions occur (Figure 3), which forces students to carry out more executions, to experiment with the tool through trial and error learning and a try-action-feedback process (Falkner, Vivian, Piper, & Falkner, 2014). The more students take action the more feedback they get from the tool. Since feedback is instant, each action (execution) provides them a new learning experience that helps them adjust and improve their code.

According to Figure 8, group work makes the most of the tool features, whereas it proves to be a more engaging, fruitful and rewarding activity. The more students use the tool and consult the tool log report, the more possibilities they have to recover from errors, improve their code and thus complete their assignment successfully. This process certainly contributes to making the tool a useful and worthwhile experience.

Conclusion and limitations

This work presented a new automated assessment tool in real online distributed programming assignments which has been evaluated at two levels: first, analysing students' own perceptions on their learning improvement and, second, exploring whether students' interactions with the tool yielded a fruitful learning experience. As shown in our review of the literature, this research

work is one of the first attempts to thoroughly examine the effects of an automated assessment tool in the area of online distributed programming. As such, we are conscious that this is the beginning of a complex and challenging endeavour and that more work still needs to be done in order to improve both the tool design and achieve a really worthwhile learning experience for both students and teachers. Limitations of the current work provide challenging opportunities for future research, such as: the generation of a more descriptive and explanatory feedback that is more adaptable to students' knowledge level, a more rigorous analysis of the log files of students' interactions with the tool using artificial intelligence methods, the use of log data analysis not at the end but during the learning process so that teachers can intervene to monitor and scaffold students' progress at critical points. Finally, our analysis showed that fruitful learning was evident only when students achieved high familiarization and constant activity with the tool. At this point, our tool should care for an efficient workload and time management, which are very important issues for online higher education students. Future work will consider an experimental research design involving an improved version of the tool with a control and an experimental group in order to provide a more thorough analysis of the values of the new tool. Moreover, another interesting issue could be establishing a categorization between those students who passed the course successfully and those who did not. Based on this, we will be able to perform a different critical analysis of the advantages and disadvantages of using our tool.

Statements on open data, ethics and conflict of interest

A request to access data can be directed to authors. All student data is de-identified and stored in the university database.

The research performed in this work is the sole work of the named authors. It has not been previously published or submitted elsewhere.

There is no conflict of interest.

References

Baker, R. S., & Rossi, L. M. (2013). Assessing the disengaged behaviors of learners. In R. Sottilare, A. Graesser, X. Hu, & H. Holden (Eds.), *Design recommendations for intelligent tutoring systems* (Vol. *1*, pp. 155–166). Orlando, FL: Learner Modeling. U.S. Army Research Lab.

Barker-Plummer, D., Dale, R., & Cox, R. (2012). Using edit distance to analyse errors in a natural language to logic translation corpus. Proceedings of the 5th International Conference on Educational Data Mining (pp. 134–141). Chania, Greece.

Becker, B. A., Glanville, G., Iwashima, R., McDonnel, C., Goslin, K., & Mooney, C. (2016). Effective compiler error message enhancement for novice programming students. *Computer Science Education*, *26*(2–3), 148–175. https://doi.org/10.1080/08993408.2016.1225464

Chen, H.-M., Chen, W.-H., Hsueh, N.-L., Lee, C.-C., & Li, C.-H. (2017). ProgEdu – An automatic assessment platform for programming courses. In Meen, Prior & Lam (Eds.), *Proceeding of the 2017 IEEE International Conference on Applied System Innovation* (pp. 173–176). Sapporo, Japan.

Clark, I. (2012). Formative assessment: Assessment is for self-regulated learning. *Educational Psychology Review*, *24*(2), 205–249. https://doi.org/10.1007/s10648-011-9191-6

Denny, P., Luxton-Reilly, A., & Carpenter, D. (2014). Enhancing syntax error messages appears ineffectual. Proceedings of the 19th ACM annual Conference on Innovation & Technology in Computer Science Education (pp. 273–278). Uppsala, Sweden: ACM.

Falkner, N., Vivian, R., Piper, D., & Falkner, K. (2014). Increasing the effectiveness of automated assessment by increasing marking granularity and feedback units. Proceedings of the 45th ACM technical symposium on Computer Science Education (pp. 9–14). Atlanta, GA: ACM.

Foley, S. S., Koepke, D., Ragatz, J., Brehm, C., Regina, J., & Hursey, J. (2017). Onramp: A web-portal for teaching parallel and distributed computing. *Journal of Parallel and Distributed Computing, 105*, 138–149. https://doi.org/10.1016/j.jpdc.2017.01.014

Gerdes, A., Heeren, B., Jeuring, J., & van Binsbergen, L. T. (2017). Ask-Elle: An adaptable programming tutor for haskell giving automated feedback. *International Journal of Artificial Intelligence in Education, 27*(1), 65–100. https://doi.org/10.1007/s40593-015-0080-x

Golding, R. (1992). *Weak-consistency group communication and membership* (Ph.D. thesis). University of California, Santa Cruz.

Grivokostopoulou, F., Perikos, I., & Hatzilygeroudis, I. (2017). An educational system for learning search algorithms and automatically assessing student performance. *International Journal of Artificial Intelligence in Education, 27*(1), 207–240. https://doi.org/10.1007/s40593-016-0116-x

Heffernan, N. T., & Heffernan, C. L. (2014). The ASSISTments ecosystem: Building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching. *International Journal of Artificial Intelligence in Education, 24*(4), 470–497. https://doi.org/10.1007/s40593-014-0024-x

Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research (pp. 86–93). Koli, FL: ACM.

Kaya, M., & Özel, S. A. (2014). Integrating an online compiler and a plagiarism detection tool into the moodle distance education system for easy assessment of programming assignments. *Computer Applications in Engineering Education, 23*(3), 363–373. https://doi.org/10.1002/cae.21606

Kelleher, J. (2014). Employing git in the classroom. In Computer Applications and Information Systems (WCCAIS), 2014 World Congress (pp. 1–4). Hammamet, Tunisia.

Lawrance, J., Jung, S., & Wiseman, C. (2013). Git on the cloud in the classroom. In Proceedings of the 44th ACM Technical Symposium on Computer Science Education (pp. 639–644). New York, NY: ACM.

Loeliger, J., & McCullough, M. (2012). *Version control with git: Powerful tools and techniques for collaborative software development*. Sebastopol, CA: O'Reilly Media Inc.

Maggi, P., & Sisto, R. (2007). A grid-powered framework to support courses on distributed programming. *IEEE Transactions on Education, 50*(1), 27–53. https://doi.org/10.1109/TE.2006.879806

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., & Silvasti, P. (2004). Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education, 3*(2), 267–288.

Mostafavi, B., & Barnes, T. (2017). Evolution of an intelligent deductive logic tutor using data-driven elements. *International Journal of Artificial Intelligence in Education, 27*(1), 5–36. https://doi.org/10.1007/s40593-016-0112-1

Pettit, R. S., Homer, J. D., Holcomb, K. M., Simone, N., & Mengel, S. A. (2015). Are automated assessment tools helpful in programming courses? In Proceedings of the 122nd ASEE Annual Conference & Exposition (pp. 1–20). Seattle, WA.

Ramos, J., Trenas, M. A., Gutiérrez, E., & Romero, S. (2013). E-assessment of Matlab assignments in Moodle: Application to an introductory programming course for engineers. *Computer Applications in Engineering Education, 21*(4), 728–736. https://doi.org/10.1002/cae.20520

Rivers, K., & Koedinger, K. R. (2017). Data-driven hint generation in vast solution spaces: A self-improving python programming tutor. *International Journal of Artificial Intelligence in Education, 27*(1), 37–64. https://doi.org/10.1007/s40593-015-0070-z

Robinson, P. E., & Carroll, J. (2017). *An online learning platform for teaching, learning, and assessment of programming, global engineering education conference* (pp. 547–556). Athens, Greece: IEEE.

Rodríguez-del-Pino, J. C., Rubio-Royo, E., & Hernández-Figueroa, Z. J. (2012). A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. In Proceedings of the 2012 International Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government (pp. 80–85). Las Vegas, NV: CSREA Press.

Rubio-Sánchez, M., Kinnunen, P., Pareja-Flores, C., & Velázquez-Iturbide, A. (2014). Student perception and usage of an automated programming assessment tool. *Computers in Human Behavior, 31*, 453–460. https://doi.org/10.1016/j.chb.2013.04.001

Stajduhar, I., & Mausa, G. (2015). Using string similarity metrics for automated grading of SQL statements. In Proceedings of the 38th IEEE International Convention on Information and Communication Technology, Electronics and Microelectronics (pp. 1250–1255). Opatija, Croatia: IEEE.

Staubitz, T., Klement, H., Renz, J., Teusner, R., & Meinel, C. (2015). Towards practical programming exercises and automated assessment in Massive Open Online Courses. Proceedings of the 2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (pp. 23–30). Zhuhai. China: IEEE.

Sukhoroslov, O. (2018). Building web-based services for practical exercises in parallel and distributed computing. *Journal of Parallel and Distributed Computing*, *118*(1), 177–188. https://doi.org/10.1016/j.jpdc.2018.02.024

Sukhoroslov, O., Volkov, S., & Afanasiev, A. (2015). A web-based platform for publication and distributed execution of computing applications. In Proceedings of the 14th International Symposium on Parallel and Distributed Computing (pp. 175–184). Limassol, Cyprus: IEEE.

Suleman, H. (2008). Automatic marking with Sakai. Proceedings of the 2008 Annual Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries (pp. 229–236). Wilderness, South Africa: ACM.

Vujošević-Janičić, M., Nikolić, M., Tošić, D., & Kuncak, V. (2013). Software verification and graph similarity for automated evaluation of students' assignments. *Information and Software Technology*, 55(6), 1004–1016. https://doi.org/10.1016/j.infsof.2012.12.005

Supporting Information

Additional supporting information may be found in the online version of this article at the publisher's web site:

Figure A1: Architecture of DSLab tool.
Figure A2: Login into DSLab tool.
Figure A3: Files uploaded by student s when submitting his/her assignment.
Figure A4: List of all the assignments uploaded by student's.
Figure A5: Submitting an assignment.
Figure A6: A completed assessment of an assignment.
Figure A7: An extract of the execution logs of level INFO and TRACE generated during the execution of an assignment.
Figure A8: Example of an application that uses LSim library to get initialization parameters, wait to start its execution and send final result.
Figure A9: Example that shows how to add a log message at any point of the code.