Augmenting the teacher's perspective on programming student's performance via permanent monitoring

Nuno Gil Fonseca
College of Technology and Management of Oliveira do Hospital, Polytechnic Institute of Coimbra
Oliveira do Hospital, Portugal
nuno.fonseca@estgoh.ipc.pt

Luís Macedo
Center for Informatics and Systems of the University of Coimbra, Dep. Of Informatics Eng., University of Coimbra,
Coimbra, Portugal
macedo@dei.uc.pt

Maria José Marcelino
Center for Informatics and Systems of the University of Coimbra, Dep. Of Informatics Eng., University of Coimbra,
Coimbra, Portugal
zemar@dei.uc.pt

António José Mendes
Center for Informatics and Systems of the University of Coimbra, Dep. Of Informatics Eng., University of Coimbra,
Coimbra, Portugal
toze@dei.uc.pt

***Abstract*— This Innovative Practice Work in Full Paper describes the use of a tool aimed to support the teacher's decisions and interventions within the scope of programming classes. As computers play an increasingly important role in our lives, the demand for programmers is growing. In addition to the intrinsic difficulties of programming, teachers must also deal with the vast diversity of students found in their classes. Close monitoring by teachers has been identified as a critical factor for student's success. However, it may be difficult for teachers to stay up-to-date at any moment about each of their students' overall progress and capabilities. For this purpose, we have developed a tool that collects snapshots of the students' source code whenever they compile it. The snapshots are processed, and a series of visualizations and aggregated data is made immediately available for teachers. Since teachers can't be permanently looking at the system, we have also developed an automatic notification system, based on cognitive theories of selective attention, which will notify teachers about numerous relevant aspects concerning their students' performance. We present the results of a field experiment, which confirms that the tool indeed allows teachers to have a deeper insight into their students' performance and progress, enabling them to make more grounded interventions.**

Keywords—programming, progress monitoring, dashboard

I. INTRODUCTION

The presence of technology in our daily life has increased so much in the last decades that, currently, several tasks simply can't be done without the assistance of technology (*e.g.*, filling the taxes). To support this proliferation of technology, we must have properly prepared professionals. However, as several reports mention [1], there is a considerable lack of professionals capable of responding to the ever-increasing demand. To cope with this demand, in the last decade there has also been a consistent increase in the number of students enrolling in computer science programs at different levels, from "boot camps" to doctoral programs. Unfortunately, the number of teachers hasn't seen the same increase.

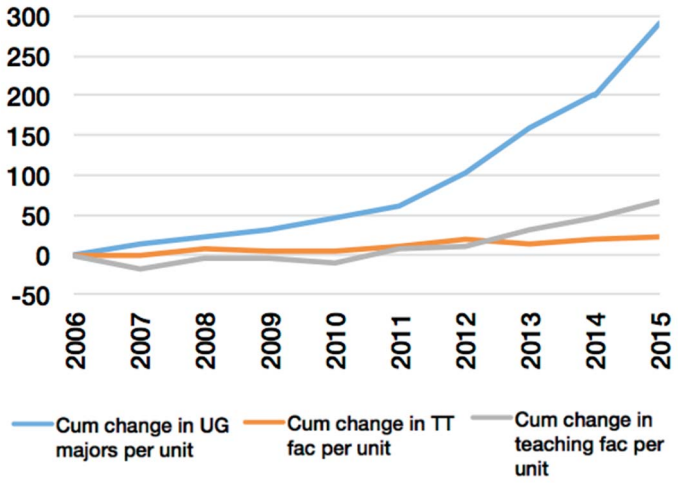

Fig. 1. Cumulative percent growth of CS majors and instructional faculty since 2006 (adapt. **[2]**).

As shown in Fig. 1 extracted from a report by the Computer Research Association [2], although there is, in fact, an increase in the number of teachers, this increase does not match in any way the growth in the number of students. As mentioned previously, the number of students enrolling in doctoral programs has also increased considerably. However, contrarily to what happened in the past, most of the Ph.D. graduates are recruited to work in the industry, rather than in academia [3].

To deal with the increase in the number of students, and a shortage of teaching personnel, several universities are forced to take actions like reduce low-enrollment courses, reduce non-major offerings or increase blended and online offerings. Some other actions that have already been taken or considered involve increasing the size of classes or the number of teaching hours assigned to each teacher [3].

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

While all these measures are needed, it is necessary to be sure that they don't compromise the quality of teaching. In the vast array of scientific areas of computer science, we are particularly interested in programming. Since programming is such an important building block of the computer science student's education, it is very important that special attention is taken concerning their experience with introductory programming courses [4].

Studies have shown that learning programming is not an easy task, and the students can experience difficulties during the learning process [5] [6] [7]. Among these difficulties, we may find different aspects like the demand for a high level of abstraction and a high capacity to solve problems, the complexity of the syntax or the difficulty to understand compile errors.

From another perspective, it is also clear that teaching programming also has its difficulties and challenges [8] [9]. Among these challenges and difficulties, we may consider mainly the following: having too many students per class; low number of contact hours with students; long list of topics to cover; topics permanently changing due to industry demand; and, dealing with plagiarism and the huge diversity among students (in terms of prior-knowledge and motivation). It is then obvious that increasing the teaching load or the classes' size *per se* might be an interesting solution from a financial point of view, but eventually counterproductive from a pedagogical perspective.

As stated by Bloom [10], the most efficient teaching-learning method is individualized tutoring, as the teacher is more aware of the students' difficulties and capabilities, and can more easily adapt the teaching pace, and the materials and assignments proposed to the students. Likewise, Raabe et al. [11] and Mendes et al. [12] claim that the students' performance is highly coupled with the ability that the teachers have to detect their difficulties and react within useful time.

Given the large number of students, this proximity relationship between teachers and students may not be fully achieved. Nonetheless, it can be increased through the use of adequate tools [5]. There are already some systems that provide some information mainly to students. However, they provide very little information to the teacher about the performance of the students. In general, they provide only information about the number of students that attempted and managed to solve each assignment or the number of assignments that each student was able to solve.

For this purpose, we propose the use of a tool able to autonomously capture information about the students' performance and provide analytics to the teacher (data and visualizations). This way, we are able to considerably increase the teacher's knowledge about their students' capabilities and difficulties and thus allowing them to make more adequate decisions.

In the next section, we present some relevant related work. We proceed by presenting details of the approach that we are proposing, followed by the main results of two field experiments with students and, finally, we make some conclusions and provide some suggestions for future improvements.

II. Related work

One important field of research related to the programming teaching-learning process is the development of Intelligent Tutoring Systems (ITS) [13], which are systems capable of assisting the students in the process of learning autonomously without or, at least, with minimal intervention of a human tutor.

Although these efforts are important, van Lehn [14] has shown that students have a higher chance of succeeding when the teaching process is conducted by a human tutor versus an ITS. It was proven that ITSs are capable of performing better in assessing/modeling the students' knowledge, but human tutors have a big advantage regarding providing feedback, clarification, and motivation to students.

Directly related to ITS, more recently, there has been a growing interest in the development of auto-graders, whose main purpose is to automatically evaluate and grade the students' tasks. Since programming involves much practice, these systems are particularly interesting, since they can save time for teachers, allowing them to focus on other aspects of the teaching process. Unlike ITS, auto-graders are supposed to be included naturally in the context of traditional face-to-face classes. As examples of auto-grading systems coming from academia, we may find Web-Cat [15], Retina [16] or TestMyCode [17]. In the last couple of years, several commercial systems have appeared as well. Examples are Voccareum [1], Codio [2], gradescope [3], Mimir [4] or CodeLab[5].

In addition to the auto-grading component, most systems also provide plagiarism detection features, mainly via the use of well-established systems, like MOSS [18].

All the systems presented previously can be valuable tools to assist students by providing information about the correctness (or not) of their solutions. Nonetheless, they have some limitations, such as being limited to a pre-defined set of assignments (e.g., CodeLab) or the complexity to prepare a new assignment (e.g., gradescope). In some cases, like with Web-Cat, the process to submit the solutions to be graded might be somewhat complicated. Another relevant aspect concerns the fact that several auto-graders imply that the students need to use a particular web-based IDE, which they may not use in their professional activities.

Particularly focused on presenting to the teacher some details about the students' performance as well as some predictions about their future performance, we may find systems like the Instructor Dashboard proposed by Diana et al. [19] or the dashboard proposed by Matsuzawa et al. [20]. However, even in these cases, the information provided to teachers is very focused on a specific aspect of the students' performance.

III. Our approach

We have developed CodeInsights, a modular web-based tool that allows teachers to obtain real-time information about their

[1] https://www.vocareum.com/
[2] https://codio.com
[3] http://gradescope.com
[4] https://www.mimirhq.com/
[5] https://www.turingscraft.com/

students' performance, no matter when or where they are working, enabling them to identify problematic situations and act to solve them.

Although not exclusive, CodeInsights was planned for a context where teachers provide one or more worksheets for each topic with a certain number of "one-function" assignments to practice on that topic. Traditionally, students don't submit their solutions for these assignments, which represents a relevant loss of information about the students' performance. In most cases, the difficulties of the students are only identified at formal evaluation moments (e.g., exams or project delivery), when there is little to do to overcome them. Overall, the system should be used in a totally formative way.

In CodeInsights, the teachers are free to decide which assignments they propose to their students and, for this purpose, the system has a tool that allows them to insert and edit the assignments in a very straightforward way. Unlike what happens with most existing systems, we have decided that students should be using the tools that they will probably use in a professional environment. So, instead of a custom web-based IDE, to use CodeInsights, the students only need to add one plug-in to the programming IDE already in use in the course. Currently, the CodeInsights students' plug-in is available for Atom, Eclipse and JetBrains IDEs (such as PHPStorm, IntelliJ or PyCharm). From the point of view of the monitoring tool used by the teacher, the IDE/editor used to develop the code is not relevant. Furthermore, as long as it is possible to use the existing API to submit the code, support for other IDE/editors can easily be added.

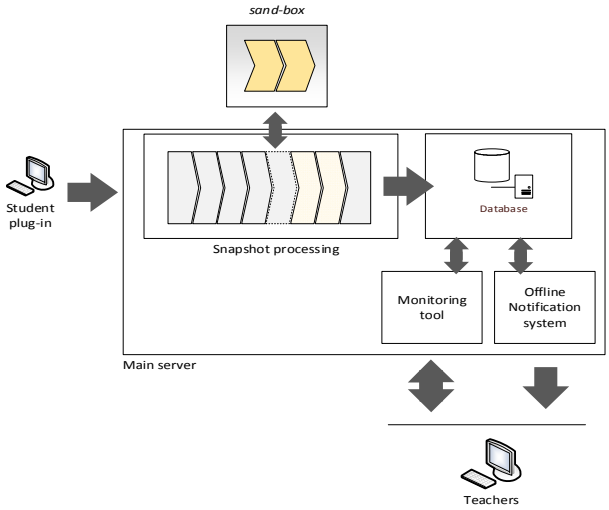

Fig. 2. CodeInsights's global architecture.

Fig. 2 shows a representation of the global architecture of CodeInsights. It takes as input snapshots of the source code that students are developing to solve a particular assignment. These snapshots are automatically sent to the "main server" each time students run their programs (in most IDEs, running a program implies that it is automatically compiled before running it).

A. *Processing the snapshots*

When the snapshots arrive at the server, the processing phase begins. First, the code is sent to be compiled/interpreted at the "sand-box server". Although this component of the system could be placed on the same server, for security and performance reasons it is better to place it on a different server.

Depending on the programming language used, the code is compiled (e.g., C or Java) or interpreted (e.g., PHP or Python). In any case, if the code has compilation or interpretation errors, it will be considered as "having compile errors" and information about the errors will be collected. If there are no compilation or interpretation errors, the code will run as is and the output will be collected. Then, the code will be tested using a set of test cases to check if the code is working as expected (*i.e.*, if the obtained output is equal to the expected one). Regardless of the code having errors or not, the results from this phase are returned to the "snapshot processing" component.

If the outputs obtained in the test cases are equal to the expected ones, the code will be considered "correct". If the outputs are not exactly equal to the expected values, but they are very similar (eventually, with additional formatting characters, or extra blank spaces), the code will be considered "potentially correct" and teachers can later decide if it is, in fact, correct or not. Finally, if the outputs are entirely different from the expected values, the code will be marked as "incorrect".

For simplicity and flexibility reasons, we are using a "black-box" unit testing approach. However, we are not disregarding the source code. For each assignment, the teacher may define a list of keywords that students cannot use to solve that particular assignment. For instance, if students are asked to write code to "manually" sort an array of unordered numbers and the programming language already has a "sort" function, the teacher may define "sort" as a forbidden keyword. If students use any of the forbidden keywords, the snapshots will automatically be considered "invalid".

If the programming language allows it, each time the code is run, CodeInsights also measures its running time. If the code takes more than a certain number of seconds to run, that code probably contains an infinite loop, or simply a very badly designed algorithm and thus it will be considered "incorrect" with an indication that the maximum running time was exceeded.

Several other details about the code are collected for each snapshot, such as the total number of lines of code, blank lines or characters. All the data collected is stored in a relational database (MySQL). From that moment on, the teacher has immediate access to all the collected information in the form of a series of visualizations and pre-processed data. Although the teachers can obtain a large set of information about their students' performance by looking at the data and visualizations, they would obviously need to be using the tool at that moment.

B. *Information available for the teachers*

According to Stephen Few [21], one particularly interesting way to present information about a dynamic system is in the form of a dashboard containing the most relevant information that should easily be perceived with an "at-a-glance" look.

For this purpose, CodeInsights provides a dashboard, in which the teachers have access to a series of indicators of the students' performance (Fig.3). From this dashboard, teachers can explore deeper and have access to more detailed information about diverse aspects of the student's work. For instance, for

each class, the teacher can view a list of students with details about their overall progress (Fig. 4) and for each student, the list of assignments that she/he have attempted to solve (Fig. 5).

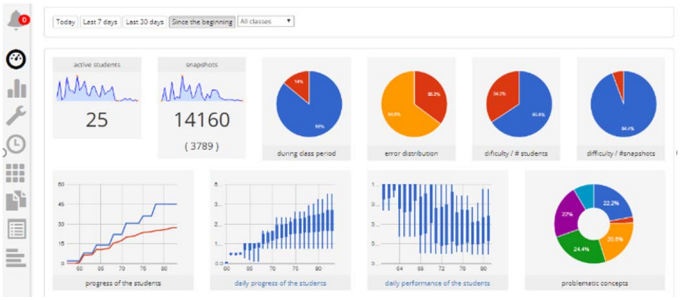

Fig. 3. The main dashboard

Fig. 4. List of students of a class with details about their overall progress

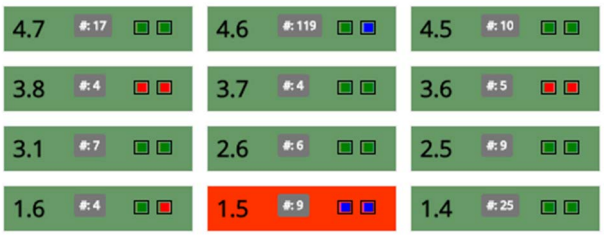

Fig. 5. A portion of the list of assignments attempted by a specific student

From the list in Fig.5, the teacher can have access to a page with details of the attempts made for each assignment, as shown in Fig. 6.

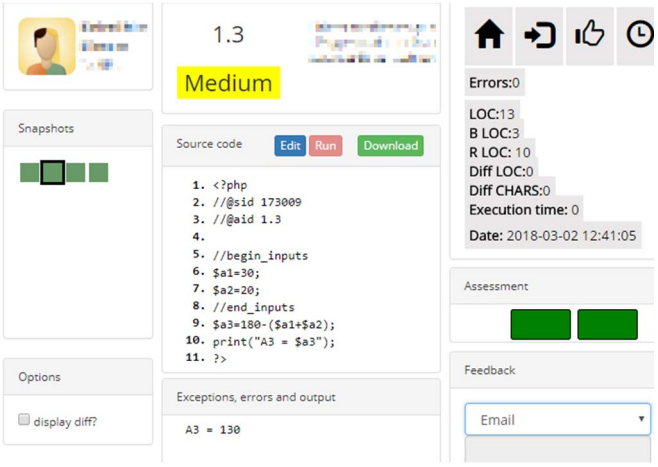

Fig. 6. A partial view of the details of a snapshot

Here, the teacher can browse through the source code of all the attempts made by that student for that assignment, consult a series of metrics (e.g., number of lines of code, execution time, number of errors), check the compilation or run-time errors, verify how the code behaved with the test cases and temporarily edit the source code and test it. Also, for each snapshot, the teacher can provide specific feedback that would be sent to the student by email along with a copy of the source code.

In this page, the teacher also has access to information about the progress and performance of students compared with the goals defined by the teacher, as well as information compared with the performance of the rest of the class (Fig.7).

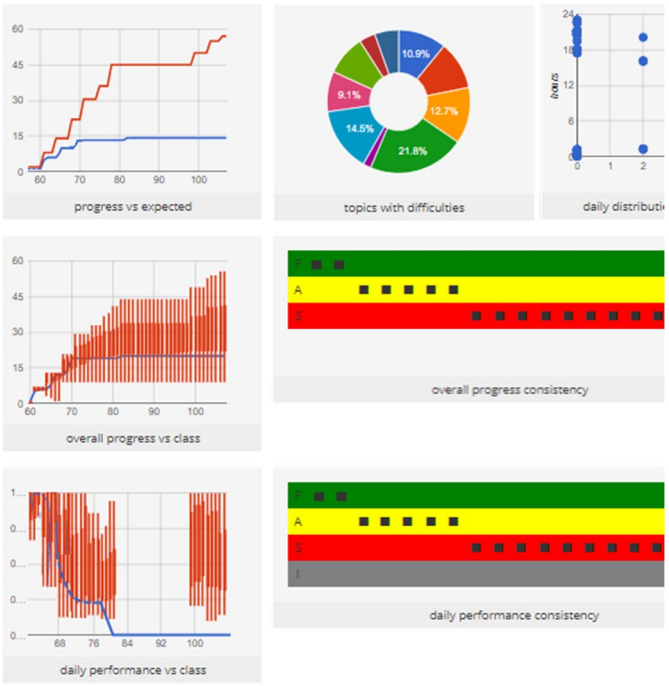

Fig. 7. A portion of the page with statistics about the performance and consistency of a particular student

Another important information that the teacher can obtain concerns the consistency of the performance and overall progress of the students. For instance, Fig. 7 shows the details of a student that initially was performing in the high (green) and average (yellow) performance areas and from a certain moment on her/his performance dropped and it ended up in the lower-performance area (red). This visualization could obviously be used to identify students whose work consistency suddenly changes dramatically, which may be an indication that the students are facing some problems that the teacher should eventually address.

The existence of students working at considerably different paces is an aspect that teachers should be aware of while teaching. Otherwise, they may simply keep moving forward without addressing the problems that some students may be facing, which in the end could result in dropout or failure. For this purpose, CodeInsights provides a set of tools that allow teachers to be aware of these differences, allowing them to take adequate measures. These tools are explained in detail in [22]. One important visualization concerning this issue is shown in Fig. 8.

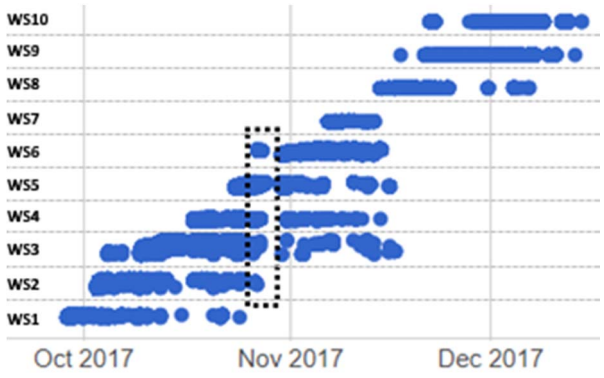

Fig. 8. Distribution of the submissions in time

In this visualization, each "dot" represents one submission, the XX axis represents the time and the YY axis represents each assignment grouped by its worksheet. The most important aspect of this visualization is the fact that there are several submissions of different worksheets at the same time. For instance, in the highlighted period, there are students working at worksheet 6, while some others are still working on assignments from worksheet 2. This information can be used by teachers to decide when it is pertinent to proceed to the next topic/worksheet as well as to determine if it is better to provide additional assistance to students in order to try to synchronize their pace.

The overall progress of the students is another important aspect that the teacher should take into account when monitoring the students' activity. For this purpose, CodeInsights offers other visualizations, like the "progress matrix" shown in Fig. 9.

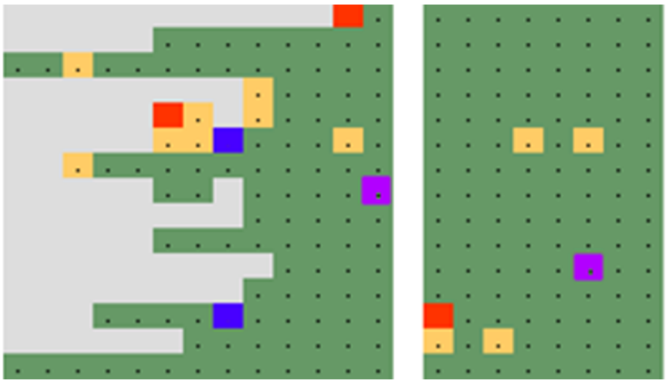

Fig. 9. A portion of the progress matrix

In this matrix, each square represents the "state" of the last submission received from a certain student (XX axis) and for a specific assignment (YY axis). The color of the squares has the following meaning: has compilation errors (red); no compilation errors, but the result was incorrect (blue); no compilation errors and the result was potentially correct (yellow); the results were correct (green); the code is invalid (purple). The grey squares indicate the student hasn't made a single attempt to solve that assignment.

Another interesting feature of CodeInsights is a live feed of students' tasks, that lets the teacher follow in real time what the students are doing, as shown in Fig. 10.

Fig. 10. A portion of the live feed

In the live feed, each line corresponds to a student, and each small rectangle represents the state of the last snapshot produced by that student in the previous *n* seconds (15 seconds in this case). The colors of the rectangles have the same meaning mentioned previously for the progress matrix (Fig. 9).

The system also provides information about the concepts that most students are having difficulties to understand (based on the assignments that the students were not able to solve correctly or haven't even attempted to solve). This information could obviously be used to reinforce the assistance provided to students concerning a certain topic via the presentation of additional material or additional, eventually simpler, assignments.

Some other visualizations allow the teacher to understand the amount of work that students need to perform outside classes, as well as in which period of the day they are working. This information can be used to eventually identify scenarios where the assignments are demanding more work outside classes that initially might have been anticipated.

C. *Automatic notifications*

Teachers, understandably, have other classes to teach as well as a plethora of other tasks to fulfill, so they most likely won't be able to be constantly looking at the data. For all these reasons, CodeInsights has another important feature that is the capability to notify the teachers about several aspects concerning the students' performance. These notifications can be generated online (in real-time when the snapshots are being processed) or offline (at a designated date and time). As examples of online notifications, we may find notifications about the excessive number of attempts to solve a certain assignment or the excessive number of consecutive attempts for an assignment with compilation errors. The offline notifications are intended to broader aspects like an unusual number of students unable to solve correctly a certain assignment. In both cases, the notification will appear in the monitoring tool as a pop-up notification (although this implies that the teacher is using the system) and by email.

We have identified many situations where the system may generate notifications to the teachers about different aspects of a student's performance. This might lead the system to generate an overload of information that the teacher may be unable to process in useful time. Because of this potential overload, we have decided to incorporate into our system one cognitive mechanism of selective attention.

In short, the system will only generate a notification if it is cognitively relevant to the teacher (i.e., if it is surprising or new)

[23]. The teacher defines expectations about the occurrence of an event in the form of probabilities. For instance, let's assume that the teacher wants to be notified when students make an unusual number of attempts while attempting to solve an assignment (more than 50). In this case, two events are possible: A, the number is lower than 50 and B, the number is equal to or higher than 50. Now let's consider P(A) = 0.95 and P(B) = 0.05. Then, depending on the event that in fact occurred, the system will check if the expectations were met (occurrence of the event with the highest probability) or not. In this particular case, in the occurrence of A, a notification certainly will not be generated since the "expected" event has occurred. In the occurrence of event B, the system will need to check how surprising the occurrence of this event was. Only if the value of surprise is higher than a predefined threshold, the notification would be generated. The selective attention mechanism is described in detail in [24].

D. *Information for the students*

Although the main focus of the system is on providing information to teachers, CodeInsights also has some features particular designed for students. One of these features is available directly at the IDE/code editor and basically consists on providing basic information about the results in the unit tests that were performed as well as the inputs and outputs used in those tests.

The students also have access to a web-based application with details concerning all the assignments that they attempted to solve. For each assignment, the student has access to details about the last attempt (source code and the results of the unitary tests performed).

E. *Privacy*

Now, more than ever, an important aspect that also must be considered concerns the ethical and privacy issues of collecting and using this type of data [25]. In the European Union, all institutions that somehow deal with personal data must be compliant with the General Data Protection Regulation (GDPR) [26]. In general terms, this document defines all the rules to collect, store and share personal data. Although these experiments took place before the GDPR came into force, in general, the rules were already being fulfilled. It is important to mention that only strictly necessary data is collected. The data that enables the teacher to identify the students consists of the name of the student, the student's id, and the student's email. Optionally, a photo of the students could also be used to help the teacher to identify them. In any case, it is very important to stress that the data collected was not used beyond what was planned in the beginning and that was clearly explained to the students. At any time, the students can decide to quit the experiment and have the data already collected completely deleted. Obviously, the data is not shared with any third-party elements.

In a previous experiment, we received the snapshots anonymized, and although it still allowed the teachers to provide overall feedback to students, it was certainly not so effective to address the particular needs of a specific student.

IV. Evaluation

CodeInsights was conceived and implemented using the Design-Based Research methodology [27]. According to this methodology, an artifact (*e.g.,* an instructional program, a textbook, a policy, or a system in our particular case) is developed iteratively, and in each iteration, a set of features is conceptualized, implemented and evaluated. Results from the evaluation processes from one iteration are assessed, and modifications to the existing features as well as new features are introduced in the next iteration.

In this paper, we have presented CodeInsights in its fourth iteration which is the result of the integration of a series of suggestions from the teachers that used the system in previous iterations, as well as some new features that we decided to include in the system. The feedback provided by the teachers regarding the current iteration of the system will certainly be used as the starting point of the next iteration.

In this iteration, CodeInsights was evaluated in two field tests in real class experiments: one introductory course on the C programming language as part of a three-year undergraduate program (experiment A); one introductory course on the Java programming as part of a one-year intensive Java programmer program (experiment B).

A. *The experiments*

In experiment A, a total of 42 students enrolled in this course, although during the period of the experiment only 29 were actually attending classes or solving the assignments. The students' involvement in the experiment was voluntary and did not have any direct influence on the final grade. During the experiment, the students should solve approximately 90 assignments divided into 9 worksheets. In the end, the system received a total of approximately 33.500 valid snapshots (of approximately 38.000 in total). The invalid snapshots were also received and processed by the system. However, most of them had problems concerning the misidentification of the student or of the assignment. Some other snapshots were actually just small code snippets used by the students to test a portion of their code alone, and not a full attempt to solve the assignment.

In this experiment, the teacher decided that he did not want to receive notifications. Nonetheless, the teacher used the other features which allowed him to follow in real-time what the students were doing and intervene whenever he felt necessary. The students used the Atom editor for writing the code and the web-based tool to follow their performance.

At the end of the experiment, the teacher was highly satisfied with the information that he could obtain from the system, and that he was not able to obtain in the past. Additionally, he manifested the intention to use the system to support his classes in the future. The students revealed that having a tool where they can have access to information about the work they had done was very interesting and after some initial reluctance on using the system, they felt that they could definitively benefit from its use.

Experiment B occurred in a less conventional context, with characteristics where the usage of CodeInsights has shown particular importance. As mentioned before, the system was used in an intensive introduction to Java course, in which the students spent 24 hours per week in class with the teachers for approximately three months.

Due to this high load, a total of 6 teachers were responsible for a certain number of hours each day. For instance, on Monday,

the students would be with teacher A for 6 hours, on Tuesday, 3 hours with teacher B and other 3 hours with teacher C, and so on. In the past, the information about where teacher A stopped in order for teacher B to be able to start was passed using a common Excel spreadsheet. The teachers would also ask at the end of each class in which assignment each student was working on. The main problem is that most students of this course are highly motivated and, although they have spent several hours in classes programming, several of them would go home and keep working for some additional hours. This means that, most times, the plan that the teachers thought they would follow was simply outdated. While using CodeInsights, the teachers could access the system, and use the several tools available to understand what was already done, the main difficulties that the students have faced, etc.

During the duration of the experiment (approximately two months), the system received more than 50.000 valid snapshots (of approximately 54.500 in total) from a total of 23 different students. The reasons for the invalid snapshots were the same as mentioned previously for experiment A. A total of approximately 130 assignments organized into 10 worksheets were offered to students. Throughout this experiment, the system has generated approximately 100 notifications concerning different aspects of the students' performance, such as the existence of students with an unusual number of attempts for a certain assignment (62 notifications), the existence of assignments without a single attempt (2 notifications), as well as the existence of students without any attempts for more than 5 days (3 notifications). Other notifications that were generated several times concerned the existence of students working at considerably different paces (35 notifications).

B. Interviews with the teachers

For both experiments, at the end of the experiment, we have conducted a semi-structured interview with the teachers. It covered several different topics, and its main purpose was to understand how the teachers have used the system and how it has allowed them to get a different insight into their student's work. Obviously, we were also interested in all the suggestions for future improvements that the teachers could provide. Next, we will present some of the most relevant questions, as well as an overview of the obtained answers.

1) How often have the teachers used CodeInsights?

All the teachers used the system while in class with the students and outside of that period as well, mainly to support the preparation of the following classes.

2) How useful was the information provided?

The information provided by the system was considered highly useful by all the teachers. Particularly, the information that the teachers were unable to obtain in another way, such as the amount of work done outside the classes.

3) Which feature of the system did you use more frequently?

The teacher from experiment A used mainly the "live feed" and the snapshot details features while in class with students and the matrix outside that period. Interestingly, the teachers from experiment B didn't use the live feed so much and said that they had used mostly the matrix both while in class with the students and outside the classes. To obtain more information about a certain assignment they also used the snapshot details feature.

4) Did you have difficulties to interpret the information provided?

The teachers mentioned that in general the information was provided in a very straightforward way. However, the system would benefit if some contextual help was introduced in certain visualizations.

5) Did you take any measures directly influenced by the information provided by the system?

The teachers mentioned that indeed they had made several interventions influenced by the system, such as addressing the difficulties that particular students were facing, provide additional time for the conclusion of certain worksheets or give extra assignments to maintain the most fast-paced students motivated.

6) In future editions of this course, or in other courses, would you consider using CodeInsights?

All the teachers that have used the system were very pleased and mentioned that they would like to use the system in the future, as they were able to benefit very much from its use. Some teachers mentioned, that they would analyze with further detail the information provided by the system to identify those assignments that put more difficulties to students, in order to understand if the assignment itself is unclear and need to be rephrased or replaced for future editions. In both cases, we were asked to allow the use of the system in the next academic year.

7) What was the reaction of the students concerning the usage of the system?

The teachers said that initially there was some reluctance in using the system, because the students didn't really understand what it was used for. However, when the teachers explained the real purpose of the system and mainly when the students started to notice that they could benefit from its use, it became natural.

8) Do you have any suggestion for future improvements to the system?

One teacher suggested that the system should allow the teacher to select which visualization should be displayed after login. Another teacher suggested that the system should allow the attribution of a certain weight to each assignment, in order to generate a "formal" grade in the end. Yet another teacher mentioned that it would be interesting to have access to information about the quality of the solution (i.e., the code may be working but not in the most efficient way). Other interesting suggestion, which has already been implemented, concerns the possibility to create a leaderboard based on certain aspects of the student's performance.

C. Discussion

The information collected in these interviews is very encouraging, as it shows that the teachers are indeed able to access a vast amount of information that they were unaware, or that was very difficult to obtain previously. However, most importantly, the teachers were able to use the information to provide better assistance to the students.

Since the teacher interventions were made directly with the students, we are not able to quantify exactly the type and number of interventions made, as well as the students that benefit from them. From the notifications generated by the system, we can understand which students were the subject of the notifications

and deduce that they were submitted to some intervention, however, we cannot be sure of that. The system itself allows the teacher to make a record of some type of interventions related to a specific code snapshot, but understandably the teachers didn't use that feature. This aspect should for certain deserve our attention in the future.

V. Conclusions

We have presented CodeInsights, a system with a set of tools that can be used to augment the perception that teachers have about their students' performance while programming.

It is particularly important to mention that the system was developed to adapt to the teacher's needs and not the other way around. This way, the use of the system should not imply any deviations from the methodology that teachers use including the IDE or code editor as well as the set of assignments provided to students. Another important aspect is the fact that the students don't need to perform any special action that they wouldn't perform if the system was not being used. Their only focus should be on solving the assignments and not on using the system.

In both experiments that took place, CodeInsights has proven very useful for the teachers to discover data that otherwise would have been missed, which has enabled them to better ground their decisions through the duration of the experiments, contributing to improve the performance of their students.

One important aspect of CodeInsights is the fact that, although it provides a vast array of tools, it allows the teachers to use only those they want. For instance, although the auto-grading feature is very useful, we were contacted by a teacher that would like to use the system without this feature in an algorithms course where he would like to correct and grade the submissions himself. The system indeed allows this.

A new version of the system is being tested which includes new features, like new notifications and visualizations, including some related to the identification of plagiarism in real-time.

One aspect that should deserve our attention involves the integration of machine learning mechanisms to support some predictions about the future performance of students in order to suggest certain interventions to teachers.

Finally, we should highlight, that due to its modular architecture, new data-sources like other programming languages and IDEs could easily be used. Also, due to the fact that all the data is stored in a relational database, we could think of a series of other data manipulations and visualizations that will be developed in the near future. Overall, we consider that the system provides a good framework for further developments.

Acknowledgment

The authors would like to thank all the students and teachers that participated in the experiments mentioned in the paper.

Financed by national funding via the Foundation for Science and Technology and by the European Regional Development Fund (FEDER), through the COMPETE 2020 – Operational Program for Competitiveness and Internationalization (POCI)

References

[1] T. Hüsing, W. B. Korte and E. Dashja, "Trends and forecasts for the european ICT professional and digital leadership labour markets (2015-2020)," 2015.

[2] CRA Enrollment Committee Institution Subgroup, "Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006," 2016.

[3] S. Zweben and B. Bizot, "2016 Taulbee Survey - Generation CS continues to produce record undergrad enrollment; graduate degree production rises at both master's and doctoral levels," Computer Research News, vol. 29 / 5, 2017.

[4] J. Bennedsen, Teaching and learning introductory programming - PhD Thesis, Norway: University of Oslo, 2008.

[5] A. J. Gomes and A. J. Mendes , "Learning to program - difficulties and solutions," in International Conference on Engineering Education, September 2007.

[6] E. Lahtinen, K. Ala-Mutka and J. Hannu-Matti, "A study of the difficulties of novice programmers.," ACM SIGCSE Bulletin, vol. 37.3, pp. 14-18, 2005.

[7] T. Jenkins, "On the difficulty of learning to program," in 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, 2002.

[8] A. Robins, J. Rountree and N. Rountree, "Learning and teaching programming: a review and discussion," Computer Science Education, vol. 13.2, p. 137–172, 2003.

[9] J. Carter and T. Jenkins, "The problems of taching pogramming: do they change with time?," in 11th Annual Conference of the Subject Centre for Information and Computer Sciences, 2010.

[10] B. S. Bloom, "The 2 sigma problem: the search for methods of group instruction as effective as one-to-one tutoring," Educational Researcher, vol. 13, pp. 4-16, 1994.

[11] A. Raabe and J. Silva, "Um ambiente para atendimento as dificuldades de aprendizagem de algoritmos," in Anais do XXV Congresso da Sociedade Brasileira de Computação, 2004.

[12] A. J. Mendes, L. Paquete, F. A. Cardoso and A. J. Gomes, "Increasing student commitment in introductory programming learning," in Frontiers in Education, Seatle, 2012.

[13] T. Crow, A. Luxton-Reilly and B. Wuensche, "Intelligent tutoring systems for programming education: a systematic review," in ACE '18 - 20th Australasian Computing Education Conference, Brisbane, Queensland, Australia, 2018.

[14] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," Educational Psychologist, vol. 46.4, pp. 197-221, 2011.

[15] S. H. Edwards and M. A. Pérez-Quiñones, "Web-CAT: automatically grading programming assignments," in ITiCSE 2008 - 13th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Madrid, Spain, 2008.

[16] C. Murphy, G. Kaiser, K. Loveland and S. Hasan, "Retina: helping students and instructors based on observed programming activities," in 40th ACM Technical Symposium on Computer Science Education, New York, NY, USA, 2009.

[17] A. Vihavainen, T. Vikberg, M. Luukkainen and M. Pärtel, "Scaffolding students' learning using TestMyCode," in 18th ACM Conference on Innovation and Technology in Computer Science Education, (New York, NY, USA, 2013.

[18] A. Aiken, "MOSS, a system for detecting software plagiarism," 2002.

[19] N. Diana, M. Eagle, J. Stamper, S. Grover, M. Bienkowski and S. Basu, "An instructor dashboard for teal-time analytics in interactive programming assignments," in LAK '17 - Seventh International Learning Analytics & Knowledge Conference, Vancouver, BC, Canada, 2017.

[20] Y. Matsuzawa, Y. Tanaka, T. Kitani and S. Sakai, "A demonstration of evidence-based action research using information dashboard in introductory programming education," in Tomorrow's Learning:

Involving Everyone. Learning with and about Technologies and Computing. WCCE 2017. IFIP Advances in Information and Communication Technology, 2017.

[21] S. Few, Information dashboard design: the effective visual communication of data, O'Reilly Media, 2006.

[22] N. G. Fonseca, L. Macedo and A. J. Mendes, "Supporting di erentiated instruction in programming courses through permanent progress monitoring," in Proceedings of the 49th ACM Technical Symposium on Computer Science Education, Baltimore, Maryland, USA, 2018.

[23] L. Macedo, "Arguments for a computational model for forms of selective attention based on cognitive and affective feelings," in 5th International Conference on Affective Computing and Intelligent Interaction (ACII 2013), 2013, 2013.

[24] N. G. Fonseca, L. Macedo and A. J. Mendes, "Monitoring the progress of programming students supported by a digital teaching assistant," in Portuguese Conference on Artificial Intelligence, Porto, Portugal, 2017.

[25] J. Sabourin, L. Kosturko, C. FitzGerald and S. McQuiggan, "Student privacy and educational data mining:," in EDM2015 - 8th International Conference on Educational Data Mining, 2015.

[26] European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council," Official Journal of the European Union, vol. L119 (6), pp. 1-88, 4 May 201.

[27] Design-Based Research Collective, "Design-based research: An emerging paradigm for educational inquiry," Educational Researcher, vol. 32(1), pp. 5-8, 2003.