Received: 7 July 2022 | Revised: 24 April 2023 | Accepted: 29 April 2023

DOI: 10.1002/cae.22640

RESEARCH ARTICLE

WILEY

An empirical evaluation of the formative feedback supported by dashboard in the context of compilation error

Miguel Sanchez-Santillan | **Carlos Fernandez-Medina** | **Juan R. Perez-Perez** | **MPuerto Paule-Ruiz**

Department Computer Science, University of Oviedo, Oviedo, Asturias, Spain

Correspondence
MPuerto Paule-Ruiz, Federico García Lorca, 18 Office 39, Oviedo, 33007, Asturias, Spain.
Email: paule@uniovi.es

Funding information
Universidad de Oviedo, Grant/Award Number: GR-2011-0040; Spanish Department of Science, Innovation and Universities, Grant/Award Number: RTI2018-099235-B-I00

Abstract

Formative feedback is one of the most recognized types of feedback in academics. However, for feedback to be effective it must be task-specific, immediate, corrective and positive. At present, automatic feedback is far from these characteristics and leaves the instructor as a fundamental agent in the teaching-learning process of programming. Faced with this scenario, the lecturer needs support tools that help them to give formative feedback in educational scenarios with many students and little time. In this paper, we have demonstrated that, by using a tool called COLMENA, any lecturer may give students effective formative feedback which is task-specific, immediate and corrective. COLMENA is a system that combines real-time Eclipse IDE data retrieval on compilation errors and analytical dashboards in a webapp solution for lecturers and students. In this study, two different approaches have been compared, considering two academic courses with and without COLMENA feedback, respectively. The results indicate that novice students receiving feedback from the lecturer via COLMENA reduced their errors, demonstrating that feedback generated from compilation errors is effective.

KEYWORDS
analytics, errors, feedback, novice students

1 | INTRODUCTION

Novice learners spend a lot of time correcting compilation errors that the development environment hands back to them [3]. For them, the environment becomes a proxy for the instructors, with students taking time to correct mistakes and learn the programming language depending on the formative feedback provided by the development environment throughout their particular learning process.

To provide formative feedback, much research has been conducted on an automatic generation [18] based on the measurement of parameters during software development. Feedback from development environments with code examples to correct errors is one of the most empirically validated guidelines. Research has shown that learners perceive the examples as an assistance to them [24, 32, 36]. Nonetheless, the feedback of the example-based development environment to the student has proven to be pedagogically questionable, as there is no guarantee that the student can relate the generic example to a specific code error or that they have sufficient knowledge to make sense of the example [3]. Furthermore, on a practical level, Prather et al. [27] noted that showing code samples to novice learners to correct errors confused them because they spent a

This is an open access article under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.

© 2023 The Authors. *Computer Applications in Engineering Education* published by Wiley Periodicals LLC.

substantial amount of time searching for that code sample in their files.

Another approach to formative feedback in development environments is that of giving suggestions or hints that offer solutions as to how errors can be corrected. This feedback has been proven to be effective when suggestions or hints are used in situations where there is a high certainty that the proposed suggestion corrects the specific error [26], otherwise the effect of "leading a user down the wrong path…" may occur [22]. Motivated in part by the disadvantages of this type of feedback, some research has opted to mix examples with suggestions or hints [32]. This second way fails in the same way as the first, that is the feedback can be ineffective because it is not task-specific [30, 33], and if the proposed solution does not correct the specific error, the feedback will not be corrective and, of course, not positive.

Recent research have included some improvements regarding feedback given by development environments, thus allowing it to be both formative and immediate [10, 20]. Despite this progress, the authors of said research have confirmed that the nature of said feedback is minimum, as it assists the programmer in correcting errors, but does not inform them of the reasons why the error occurs, which turns it more difficult to be construed and understood [3]. As a consequence, feedback is not completely formative because it does not become task-specific, corrective, and positive [30, 33]. In fact, this feedback will be determined by the quality and quantity of relevant information provided by said environment. This relevant information is obtained when the environment detects an error in the code, accesses the contextual information of the error and the context in which it has been found. From this relevant information, the environment generates feedback based on examples or hints, and we have already seen the drawbacks of both approaches.

Thus, automatic feedback is still far from being sufficiently mature. In the meantime, the lecturer remains a fundamental agent in the teaching–learning process of programming. Still, the instructor has to cope with teaching error correction in a classroom with many students and little time to do so. In most cases, their feedback is based on their years of teaching experience. It is worth remembering that novice instructors are inexperienced and provide feedback based on their experience as students or professionals, not as lecturers. Faced with this reality, they need support tools to give formative feedback to their students. With this in mind, we have developed COLMENA, which offers feedback based on two concepts:

- Analytics of the errors made by students. Through dashboards, the instructor can find out which are the most common errors in each session, which user has made them, and how many times they have appeared.
- Examples of errors in the code with their solution and suggestions on how to solve them, as well as the cause behind the error.

As opposed to automatic feedback, our work focuses on enhancing the instructional feedback of the lecturer with support tools such as COLMENA. Using the analytics, the lecturer is aware of the most common errors in the session and which user has committed them, and can therefore specifically explain the error and its cause, allowing for immediate, task-specific and contextualized formative feedback. The examples of the errors in the code with their solution, together with suggestions or hints on how to solve them, enhance the corrective and positive nature of the feedback. We therefore hypothesize that a support tool will reinforce the formative feedback given by the lecturer, allowing it to be more effective.

To corroborate this hypothesis, the research methodology we apply is a case study of novice students enrolled in a Computer Science degree. Specifically, we compare two academic years. During the first year, the same instructor gives formative feedback based on their own experience, and in the second year they rely on COLMENA to give formative feedback. The results demonstrated that tools such as COLMENA enhance the lecturer's formative feedback on students and errors.

To achieve the proposed contribution, it is necessary to know the effect of the lecturer's formative feedback with COLMENA on the students, although we also want to assess this effect on the errors. That is why we pose the following questions:

1. Per error message, is the feedback effective?
2. Per student, is the feedback effective?

The rest of the paper is organized in the following way. The following section details the related work and, in Section 3, we define COLMENA. The work methodology is included in Section 4. Section 5 includes the results and Section 6 includes their discussion. Conclusions and future work are included in Section 7.

2 | RELATED WORK

COLMENA's formative feedback is sustained by analyses of the errors that students make with the development environment, examples of errors in the code with their solution, and suggestions on how to resolve them, as well as the cause that produces the error. Hence, the state of

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

the art is focused on research that offers error analysis as well as studies based on the generation of formative feedback.

2.1 | Error analysis

There are systems that offer error analytics in different formats (i.e., dashboards). We will classify them into commercial and academic tools. Within the commercial ones, we consider two representatives and related to our research: Rollbar (https://rollbar.com) and Newrelic (https://newrelic.com/). Both tools offer error classification, error tracking, the ability to analyze a number of occurrences or even generate conversations to carry out solutions. The analysis of these tools allows us to state that they are very powerful and helpful for the software development process, but they are not well suited for novice students due to their complexity when it comes to showing and explaining both the error and the solution to it.

It is in the academic field where we find tools that are best aimed at novice learners. These tools have evolved over time. In this way, initial works, such as those developed by Jadud [16] or by Luke [21], are contextualized in a specific university and are plugins on a development IDE. The DevEventTracker [18] is a software system that interfaces with existing Web-CAT services to track student development data continuously, without any student effort. Development and compilation events are tracked within the Eclipse IDE through a plugin and sent to a Web-CAT server. The system provides a dashboard as a set of instructor-visible web pages that display useful data in generated charts and tables. Data are presented in both class overview and individual student summaries. Jadud [16] has undertaken a quantitative, empirical analysis of introductory programmer compilation behaviors using an extension to BlueJ. Jadud is more focused on compilation events while DevEventTracker tracks event data. In addition, DevEvenTracker monitors students in the development of projects and Jadud focuses on student sessions with BlueJ.

Recently, a more detailed approach of error analysis carried out by BlueJ is included in Karvelas and Becker [17], in which the authors compare the error analysis and production made by BlueJ 3 with that of BlueJ 4. In said work, the authors have underlined the importance of automatic compilation of Blue J4 against the manual one of BlueJ3. They said that automatic compilation gives the possibility to show errors to students one by one based on the compilation made by the environment. This approach has the advantage of avoiding students' overloading with many errors. Nevertheless, automatic compilation has the problem that the error may be due to the fact that the student has not finished writing the sentence.

These previous works lead to systems that are more focused on large-scale data collection from heterogeneous sources. For instance, Blackbox [4] is a perpetual data collection project that collects data from worldwide users of the BlueJ IDE-a programming environment designed for novice programmers. The collected data is anonymous and is available to other researchers to use in their own studies, thus benefiting the larger research community. Since 2018, it has recorded more than 306 million build events, including all error and warning messages. The authors, in the 19 published papers, propose examples of data analysis, highlighting in the same vein as Jadud, the top 10 most frequent compilation errors as a feedback tool.

COLMENA is aligned with Jadud's or Luke's approach and we depart from BlackBox's philosophy. We believe that large-scale data processing is appropriate for making predictions associated with performance, but not for providing feedback that is task-specific, corrective, positive, and immediate. Jadud's or Luke's approach to collecting and processing student events in the IDE is the basis of COLMENA, but in this case, our system is more associated to the error-specific concept, that is, it collects the error associated to a session in which the lecturer delivers a specific content at a specific time, not to an Eclipse project that the lecturer analyzes afterwards.

2.2 | Automatic formative feedback

As we noted in the introduction, the automatic formative feedback is primarily based on code examples to correct errors and on suggestions which provide solutions as to how the errors can be corrected.

Regarding examples of error-correcting code, research is inconclusive on this type of feedback. Denny et al. [8] developed a system that provides enhanced error messages (ECMs), including concrete examples that illustrate the kind of error that has occurred and how that type of error could be corrected. They evaluated the effectiveness of the enhanced error messages with a controlled empirical study and found no significant effect. In contrast, Becker [2] similarly enhanced error messages in the automated assessment tool, Decaf, also used for Java programming. His findings showed that these enhanced messages did change student behavior. After viewing an enhanced error message, students were less likely to generate the same error in the future. Pettit et al. [25] enhanced compiler error messages in an automated tool, Athene, used for C++ programming.

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

They could not find conclusive results that the enhanced compiler error messages were more helpful than standard compiler error messages. Finally, recent studies such as the one made by Denny et al. [9] have shown the difficulty novice students have when understanding and construing the ECMs, considering error messages as notoriously problematic, due to their lack of readability.

The formative feedback provided by suggesting solutions or hints has a problem, it fails to be task-specific, that is, it fails to determine whether the proposed suggestion corrects the specific error. This problem has been tackled by different techniques [14, 36] with unsatisfactory results. In this way, HelpMeOut [14] is a social recommender system that aids the debugging of error messages by suggesting solutions that peers have applied in the past. Hartmann et al. [14] conducted an evaluation with novice students and concluded that the system can suggest useful fixes for 47% of errors after 39 person-hours of programming in an instrumented environment. BlueFix, an online tool currently integrated into the BlueJ IDE which is designed to assist programming students with error diagnosis and repair [36]. BlueFix suggests methods to resolve syntax errors using a database of crowd-sourced error fixes. They have conducted an evaluation of BlueFix, which revealed that 11 students viewed the tool positively as a support to help correct errors, and an initial evaluation of BlueFix precision suggests an improvement of 19.52% over HelpMeOut system [14].

Recently, a third way of dealing with feedback has appeared, based on mixing code examples to solve compilation errors with suggestions or hints [32]. Hence, Thiselton and Treude [32] developed two software tools, one to augment errors with formal documentation providing examples of the correct use of a Python feature, and one providing suggested solutions to identify errors using highly ranked responses on Stack Overflow. Thiselton and Treude's evaluation involves 16 participants (14 professional programmers and 2 students whose level is not indicated). Most of the participants rate Stack Overflow's answers very positively, but it is worth bearing in mind that it is a very small sample and basically made up of professionals who already have an advanced understanding of programming errors.

Using the advantage of large-scale data collection with tools such as MOOCS, works have emerged addressing the challenge of providing fully automated, personalized feedback to students for introductory programming exercises without requiring any instructor effort [35]. It has been tackled using the following techniques: formal methods, programming languages, and machine learning. Most of these techniques model the problem as a program repair problem: repair an incorrect student submission to make it functionally equivalent. The result, in most cases, is that these tools are not very effective, impractical and, to achieve good results, a minimal repair is required. Other techniques are based on error detection and solution [1]. Nevertheless, as the authors say, "correctness of a fix that differs from a user's fix requires manual examination, and it is not practical to do it at a large scale." Finally, the large-scale application of hint generation is proposed by Phothilimthana and Sridhara [26]. The advantage of this approach is that the vast amount of data allows them to apply different techniques to the generation of hints, offering up hints for any programmer and no matter how far away from a correct solution it is.

None of the three ways of providing automatic formative feedback have solved the problem of feedback being task-specific, let alone corrective or positive. The primary reason lies in a twofold technical difficulty: On the one hand, the generation is neither efficient nor practical, and on the other hand, there is a difficulty in correcting an error or generating an appropriate suggestion or hint because it is difficult to provide an appropriate context for the error [23]. Nevertheless, the student needs the formative feedback to be able to continue with his or her learning on programming, and development environments are not able to provide such feedback because, from a technical point of view, they do not provide an error analysis focused on the context of the specific task within which error takes place. This technical impossibility does not provide effective formative feedback because both the examples as well as the suggestions or hints are not contextualized with the specific error. Taking into account this situation, the responsibility of the teaching and learning process of programming falls on the lecturer who must frequently work with groups made up of several students with a learning process that is specific for each student. Therefore, it is necessary for the lecturer to have supporting tools, such as COLMENA, that strengthen the task-specific and contextualized formative feedback, allowing it to be corrective and positive as well. Besides, formative feedback must be effective and this is the reason why we have compared two academic courses given by the same lecturer with the only difference that, during one academic course, said lecturer used COLMENA as supporting tool and, in the other course, this lecturer did not use any tool that might strengthen the feedback given in class.

3 | COLMENA: SUPPORT TOOL FOR LECTURER'S FEEDBACK

The COLMENA System (colmena means hive in Spanish) is the system developed to provide an answer to our research questions and constitutes the support tool

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

used by the lecturer to provide feedback. It has been developed as a plugin on Eclipse to collect compilation events along the same lines as previous works such as those of Luke or Jadud. Furthermore, it is based on the principles of learning analytics [11] and on the perspective of HCI [34]. In so doing, the analytics allow the lecturer to be aware of the situation of the students and the HCI approach suggests that to be most effective, compiler feedback should be divided into three increasing levels of elaboration: 1. Provide the programmer with a short message about the problem. Provide brief explanations or examples. 3. Provide a further level of support based upon potential corrective actions. COLMENA is composed of a set of applications that interact with each other and form part of the system architecture:

- COLMENA Plug-in: A plug-in for the Eclipse IDE that identifies, detects code changes, and collects errors produced during compile time, for later storage in a persistent environment (relational database or file system).
- COLMENA Management: A content manager that allows you to create, delete, and edit the subjects, their practice calendars and their topics. It also facilitates the association of the errors captured in the practical sessions of the courses.
- COLMENA Platform: A web application that provides real-time feedback on the errors stored by the COLMENA plug-in and which has been developed with the aim of supporting the immediacy component associated with the feedback to ensure its effectiveness.

These three systems mentioned above are involved throughout the entire learning process of programming (Figure 1). This process starts with a student attending a programming practice class to develop code and ends with the feedback the lecturer provides to students from the errors generated in that practice class.

3.1 | COLMENA plug-in

The life cycle of the plug-in begins when the user, a student in our case study, starts to develop code in the Eclipse IDE and saves or compiles the source code of the program. In this "edit-compile" cycle, the plug-in is in charge of collecting the existing errors, analyzing them to see if they should be stored and, if so, using the selected persistence system (database or log files).

COLMENA analyzes sets of successive compilation pairings for each file that a student attempted to compile during a session [37], counting the appearance of new errors that occur in the code blocks. To do so, the plug-in includes a cache (COLMENA tree) that goes through the file and determines whether a fragment of the source code has been modified as compared to its previous version (Figure 2). If an error appears in the code block for the first time, it is counted and associated with an

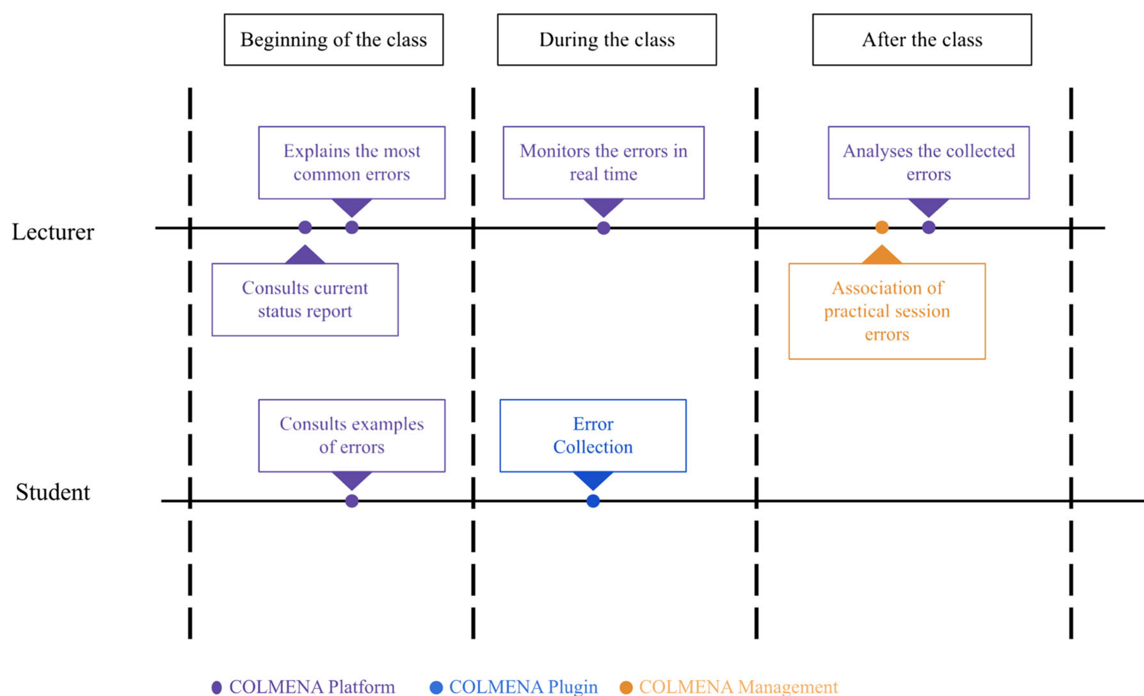


FIGURE 1 Timeline representing student or lecturer interactions with the different applications within the COLMENA System.

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

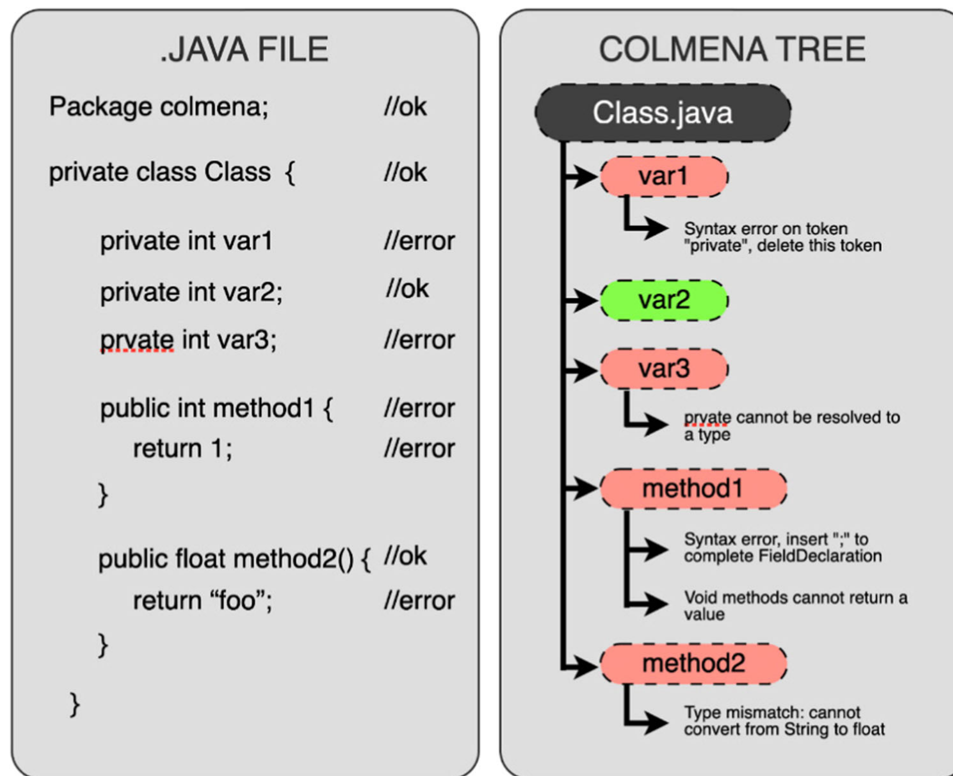


FIGURE 2 Visual representation of a Java class and the corresponding mapping that COLMENA Tree performs on the code.

error message obtained from Eclipse. In successive compilations, if the same error occurs again in the same code block, it is not counted, but it will be taken into account if it appears in another code fragment or if the original block has been modified. This approach helps the lecturer to identify which errors the students are trying to solve during the session. The advantage of the COLMENA cache is that it favors a targeted approach to the specific error associated with the context in the code.

3.2 | COLMENA management

The purpose of this application is to define the set of subjects, timetables and available sessions, automatically locating the errors shown by the plug-in in the relevant subject and session, according to the date and time detected.

3.3 | COLMENA platform

The COLMENA platform is the application in charge of providing the lecturer with feedback on the errors collected by the plug-in in real time. The purpose of the application is that the lecturer, during and after a practice session, can evaluate the typology of the errors to be able to explain in class and in subsequent sessions, if necessary, the causes and reasons underlying the error. In view of this, we suggest as an example the following scenarios of use where the lecturer can take advantage of the benefits provided by the tool:

1. A repository of bugs that can be carefully analyzed from one session to the next. In this scenario, if there has not been enough time to solve all the errors in one session, for whatever reason, it is possible to start the next session by solving those errors in the following practice class.
2. Have an up-to-date situation of the errors in each practical class. In this scenario, the lecturer is able to see the most common mistakes and solve them. If there were specific errors, they could already deal with them in a particular way for each case of student-error.

In both scenarios, the context of the code where the error occurs is present, which makes the feedback specific-error, immediate, positive and corrective. Along the lines of Prather et al. [27] who demonstrated the effectiveness of feedback tools outside the context of development tools, COLMENA is developed as a web platform and not as an integrated system within the IDE. Thus, the platform is a web application, accessible via username and password, and offers different screens with information (on subjects, practice sessions, errors,

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 1 Summary of functionalities offered by COLMENA Platform.

Action
Information about the subject
Visualization of general information about the course.
Visualization of the sessions that make up the course.
Visualization of the top of errors.
Visualization of the students of the course
Information about the practice session
Visualization of general information about the session.
Visualization of the session's error information.
Comparison of two sessions simultaneously.
Visualization of students in the session.
Information about the students
Visualization of the subjects in which they participated.
Visualization of the 10 most frequent errors of the student.
Visualization of the three most frequent errors of the student in the practice class.
Visualization of the three first errors of the student in the practice class.
Information about the errors
Visualization of information about an error and its examples.
Creation of a new example.

examples, student profile) depending on the type of user accessing it (Table 1). For the students, the 10 most frequent student errors are visualized in the same line as Jadud [16] and Luke [21]. In addition to this, the first three and the three most frequent errors are displayed. The number 3 is chosen so that the lecturer is able to remember them following the research proposed by Gobet et al. [12].

4 | METHODOLOGY

It is therefore necessary to prove that our proposal to enhance the lecturer's formative feedback with support tools such as COLMENA is really effective for both students and errors. To do so, we follow a control group–experimental group comparison that allows us to address our hypothesis and its associated research questions.

4.1 | Sample

The present study was conducted on a sample of 155 students. The sample was obtained from novice students enrolled in a Computer Science degree for two academic years. The distribution of students between the years is 77 students analyzed in the first year and 78 in the second one. Among other things, the learning outcomes of the subject are for the students to acquire knowledge and skills in the implementation techniques of algorithms such as Divide and Conquer (D&C), Backtracking (BT), Dynamic Programming (DP), or branch and bound. In relation to the number and duration of the practical sessions, four practical sessions of 120 min each, common to both courses, were taken as a reference. In the present study, the four sessions selected were: D&C, Greedy Algorithms (Greedy), DP, and BT.

To establish the study groups, all first-year students are part of the control group and the COLMENA tool was used exclusively to gather the compilation errors that were generated in each of the practical sessions analyzed. The feedback provided by the lecturer was based on maintaining a dialog with the students during the session to find out how they were making progress during the session and answering any questions raised by them, but in no case any information provided by COLMENA was used. At the end of the first year, a total of 5136 errors were gathered, related to 40 different compilation error messages. Using this set, a subset of 4389 errors (85.46% of the total) was selected, related to 18 error messages common to all 4 practice sessions. The information about these 18 errors (Table 2) provided the knowledge foundation for the lecturer, used as a fundamental tool to generate feedback for students during the second year.

The second academic year was defined as an experimental group and the same working conditions were maintained as in the previous year: the practical classes were taught by the same lecturer; the same theoretical and practical content was used; and the same four sessions were analyzed as in the previous year. The only exception was the approach to providing feedback. In this second course, the lecturer used the information generated from the selection of the errors found during the first year.

4.2 | Feedback procedure

The process of incorporating and providing feedback was the fundamental difference between the second and the first year. Using COLMENA, the procedure followed consists of two phases: before the start of the experimental group's course and the actual practice session.

4.2.1 | Before the start of the experimental group course

From the information obtained in the control group, the 18 errors are added to the COLMENA Platform,

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 2 The code field is used throughout the document to refer to a specific error message.

Code	Error message	Compilation errors (control group)
CE_1	[PLACEHOLDER] cannot be resolved to a variable.	1337
CE_2	Syntax error, insert [PLACEHOLDER] to complete ReturnStatement.	406
CE_3	[PLACEHOLDER] cannot be resolved to a type.	403
CE_4	The method contains ([PLACEHOLDER]) in the type [PLACEHOLDER] is not applicable for the arguments ([PLACEHOLDER]).	368
CE_5	Type mismatch: cannot convert from [PLACEHOLDER] to [PLACEHOLDER].	295
CE_6	The declared package [PLACEHOLDER] does not match the expected package [PLACEHOLDER].	280
CE_7	Cannot make a static reference to the non-static field [PLACEHOLDER].	229
CE_8	This method must return a result of type [PLACEHOLDER].	174
CE_9	Syntax error on token [PLACEHOLDER], delete this token.	159
CE_10	Syntax error on token [PLACEHOLDER], Statement expected after this token.	151
CE_11	The constructor [PLACEHOLDER] is undefined.	148
CE_12	Syntax error on token [PLACEHOLDER], [PLACEHOLDER] expected.	114
CE_13	The local variable [PLACEHOLDER] may not have been initialized.	87
CE_14	Duplicate local variable [PLACEHOLDER].	84
CE_15	Cannot make a static reference to the non-static method [PLACEHOLDER] from the type [PLACEHOLDER].	63
CE_16	Syntax error on token(s), misplaced construct(s).	35
CE_17	Syntax error on token [PLACEHOLDER], invalid Type.	34
CE_18	Return type for the method is missing.	22

Note: Compilation error messages common to the four practice sessions performed by the control group (first year) and whose feedback was applied to the experimental group (second year).

completing the necessary information to understand the reason why the error appears: compilation error message produced by the Java compiler, comprehensive description of the error, references to additional bibliography on the error and the subject of the error (types, variables, constructors, syntax, structural, methods, or import).

For each error message (18) and for each session (4), the lecturer generates an explanation on why the error occurs, a suggested solution to the error and examples of the Java code (2) that produces the error in question, as well as the code proposed to solve the error, resulting in 144 examples. The decision to generate this specific information in each session it is explained because the feedback must be specific-error to be corrective and positive, therefore, it needs to be as closely related to the context of the specific practice as possible. We do understand that this way of generating the error is very manual, yet we would like to point out that the lecturer only does it once and that, in case more errors appear, they will be in small numbers.

4.2.2 | At the beginning of each experimental group session

The lecturer presents, using the COLMENA Platform through the projector, how they occur and how a possible solution to the most frequent errors for that particular session can be found, with specific examples of implementation. It is important to note that the lecturer knows which are the most frequent errors in each session from the experience in the control group. If the student demands it, the lecturer assists him/her individually, in the same way as they did during the first year, but using the errors and the explanation of the COLMENA Platform.

To exemplify how feedback is incorporated in a class, we explain below the case of Miguel, a student in the experimental group. In the practice session, the lecturer introduces the objectives of the session as well as a script of the practice to be carried out and an explanation of the errors of the previous session with the support of COLMENA. After this explanation, Miguel starts programming. Additionally,

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

COLMENA project realises analytics over errores providing useful information about programming learning

ES | EN

HOME SUBJECTS USERS ERROR FAMILIES WELCOME, *lecturer* (LOGOUT)

ERROR

% CANNOT BE RESOLVED TO A TYPE

Times that appear: 14,405 Users who have: 498 Subjects where appear: 8

EDIT

This error occurs when using a type that does not exist when declaring a variable. It is quite common that this error appears because there is a mistake writing the type, but it is also the case that it appears when you try to use a complex type (such as a previously created class or a class from a library) that does not really exist.

[BARNES07]
2.3 Fields, constructors and methods
2.13 Local variables

CAUSES AND SOLUTIONS OF THIS ERROR

+ Add new example

EDIT

CODE EXAMPLE WHERE ERROR APPEARS	RIGHT USAGE WHERE ERROR DOES NOT HAPPEN

`public class App`
`{`
`private LinkedList var;`
`private itn number;`
	`public class App`
	`{`
	`private java.util.LinkedList var;`
	`private int number;`

EDIT

CODE EXAMPLE WHERE ERROR APPEARS	RIGHT USAGE WHERE ERROR DOES NOT HAPPEN

`public class Principal`
`{`
`Person persona;`
`...`
	`import modelo.Person;`
	
	`public class Principal`
	`{`
	`Person persona;`

OUTPUT FOR THIS WRONG CODE	WAY TO SOLVE THE ERROR	EXAMPLE EXPLANATION
Person cannot be resolved to a type	*Use an import sentence*	*If we use a class belonging to another package, we should previously import the package to be used.* *Otherwise, the compiler won't detect this class*

FIGURE 3 Screen with example of error displayed by the lecturer at the beginning of the practice session.

the monitoring of the students indicates that Miguel makes errors, such as "Cannot be resolved to a type" (a very frequent error concerning types). The lecturer, using the error file in the COLMENA Platform (Figure 3), explains the examples in which this error might occur. This information, new to Miguel, helps him to understand the reason behind the error, with the aim of generating it on fewer occasions, and in the case of doing so, to know how to interpret the error message in the IDE. Furthermore, the lecturer can access this error and any other error documented on the platform whenever they want to explain its solution (Figure 4).

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

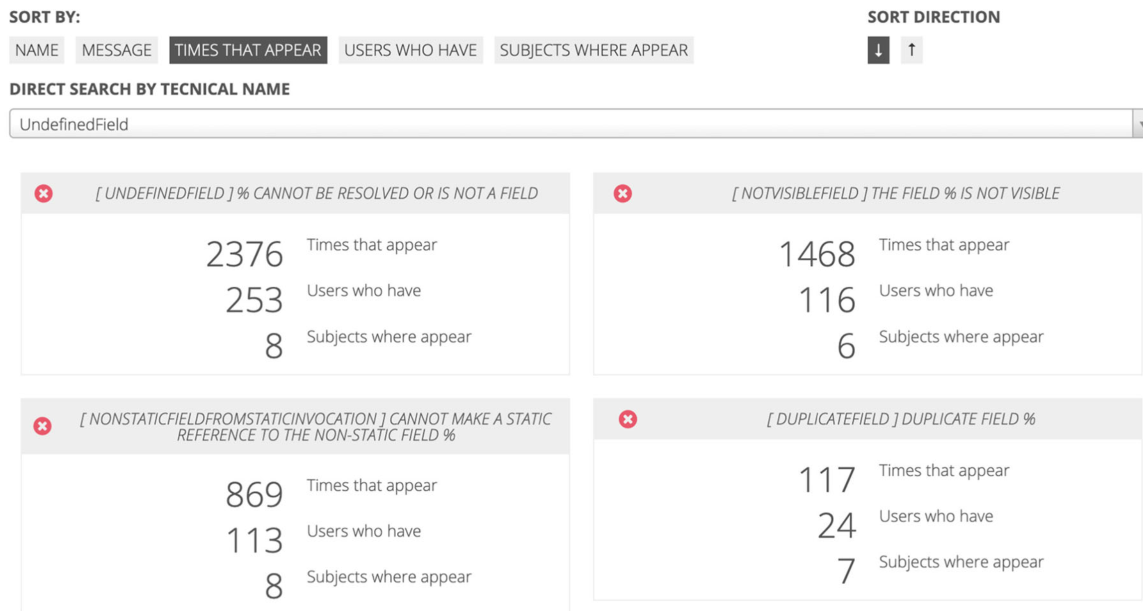


FIGURE 4 List of errors of a specific type that the lecturer could consult.

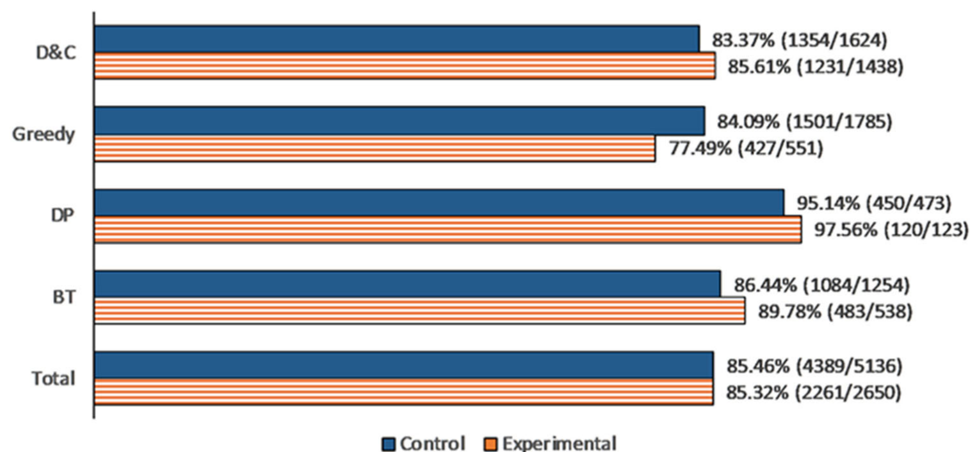


FIGURE 5 Percentage coverage of compilation error messages for the control group and the experimental group, both for each session and in total. The number of compilation errors covered out of the total is shown between brackets.

4.3 | Data analysis

In the present study, the COLMENA tool gathered 7786 compilation errors, grouped into 40 error messages. In the control group, 5136 compilation errors were gathered, while 2650 compilation errors were gathered in the experimental group. The 18 compilation error messages on which feedback was provided represent 85.41% of the total number of errors made by the students, 2261 for the experimental group and 4389 for the control group. The volumetry of compilation errors made for each of the groups in each session, as well as those covered by feedback, is presented in Figure 5.

To answer the research questions, we used statistical methods with SPSS v26. For data that can be paired, we used the Wilcoxon-rank test. For unpaired data, we used the Mann–Whitney *U* tests. For all cases we used two-tailed tests with an α level of .05 indicating significance. The effect size was calculated for all tests, expressed as r [28]. The interpretation of the size has been done following the criteria established by Cohen [5, 6], who said that: $r = .10$ is a small effect size, $r = .30$ is a medium effect size and $r = .50$ is a large effect size.

5 | RESULTS

Figure 6 shows the distribution of each of the 18 error messages studied for both the control and experimental groups. A preliminary glance seems to indicate that the distribution of the error messages is very similar for both groups, finding the maximum variation in CE_2 with 2.14%.

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

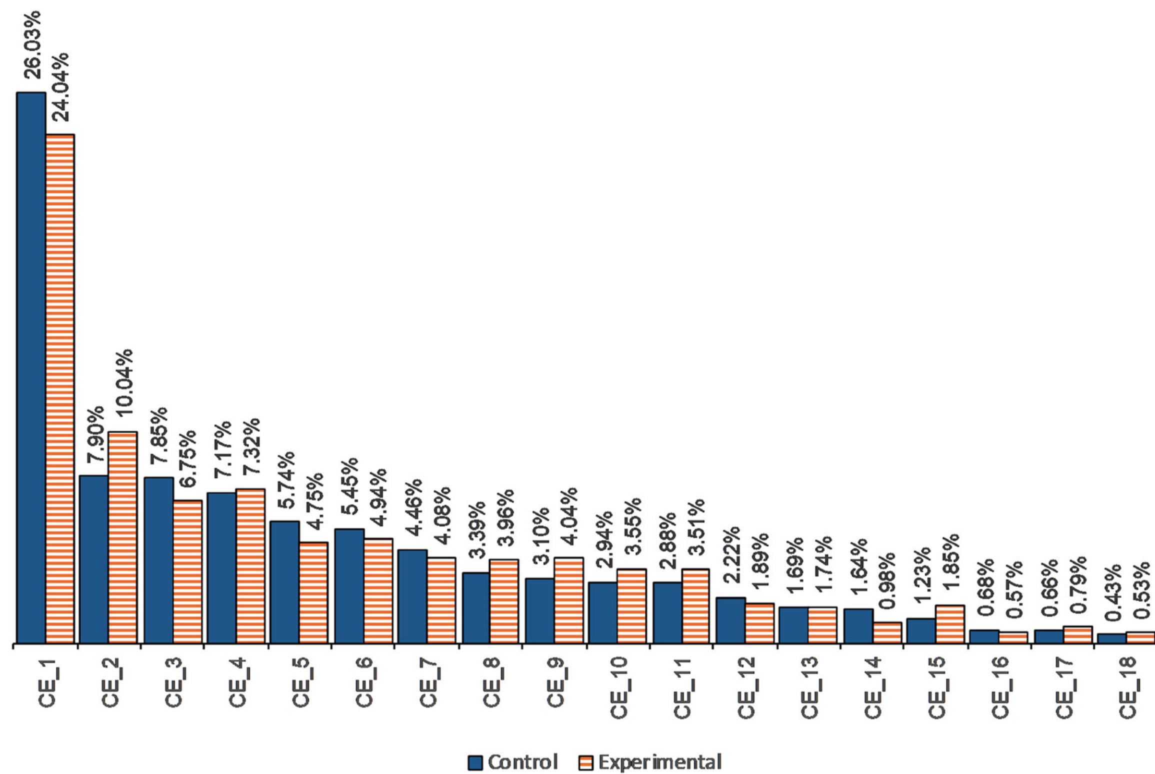


FIGURE 6 Distribution of the 18 error messages studied for both the control and experimental groups at completion of the case study.

According to the 18 error messages studied, Table 3 presents the descriptive statistics of the compilation errors collected for each group, both for each session and in total. In the same way, Table 4 presents the descriptive statistics of the compilation errors per student for each group, for each session and in total. The descriptive statistics of the compilation errors made by the students in each error message (from CE_1 to CE_18), both in total and in each session, are presented in the Appendices S1.

5.1 | RQ 1: Per message error, is feedback effective?

The Wilcoxon signed-rank test (two-tails) showed that the experimental group (median [Mdn] = 99.50) made significantly fewer compilation errors than the control group (Mdn = 155), $z = -3.73$, $p < .001$, large effect size ($r = .62$).

Next, we studied whether the feedback was effective in reducing the total number of compilation errors in each of the sessions in an independent way. At this point, it is worth remembering that the feedback was applied to the 18 compilation messages common to all sessions.

TABLE 3 Descriptive statistics of the compilation errors for each session.

	Mean	SD	Median	Min.	Max.
Total					
Control	243.83	300.6	155	22	1337
Experimental	125.61	144.59	99.50	14	637
D&C					
Control	75.22	83.25	46.50	2	363
Experimental	68.39	71.63	60.50	2	319
Greedy					
Control	83.39	98.38	53	7	424
Experimental	23.72	24.61	17.50	2	88
DP					
Control	25	35.55	16.5	4	155
Experimental	6.67	6.80	4	0	23
BT					
Control	60.22	89.80	34	3	395
Experimental	26.83	51.06	15	2	224

Abbreviations: BT, Backtracking; D&C, Divide and Conquer; DP, Dynamic Programming; Greedy, Greedy Algorithms; Max., maximum; Min., minimum.

TABLE 4 Descriptive statistics of compilation errors per student.

	Mean	SD	Median	Min.	Max.
Total					
Control	56.27	49.58	45.5	2	267
Experimental	29.36	28.88	20	0	123
D&C					
Control	17.36	18.22	13	0	67
Experimental	15.99	18.66	11	0	111
Greedy					
Control	19.24	25.71	10.5	0	120
Experimental	5.55	7.89	1	0	40
DP					
Control	5.77	8.88	1	0	42
Experimental	1.56	4.46	0	0	21
BT					
Control	13.9	21.33	5.5	0	120
Experimental	6.27	15.59	0	0	99

Abbreviations: BT, Backtracking; D&C, Divide and Conquer; DP, Dynamic Programming; Greedy, Greedy Algorithms; Max., maximum; Min., minimum.

In the first session (D&C), no statistically significant differences were found in the number of compilation errors on which feedback was applied between the experimental group (Mdn = 60.50) and the control group (Mdn = 46.50), $z = -1.138$, $p = .25$. Therefore, in this first session, students in both groups behaved similarly. In the second session (Greedy), the Wilcoxon-rank test (two-tail) found that the experimental group (Mdn = 17.50) made significantly fewer compilation errors than the control group (Mdn = 53), $z = -3.681$, $p < .001$, with a large effect size ($r = .61$). In the third session (DP), the Wilcoxon-rank test (two-tailed) found that the experimental group (Mdn = 4) made significantly fewer compilation errors than the control group (Mdn = 16.50), $z = -3.681$, $p < .001$, with a large effect size ($r = .61$). In the fourth and final session (BT), the Wilcoxon-rank test (two-tail) found that the experimental group (Mdn = 15) made significantly fewer compilation errors than the control group (Mdn = 34), $z = 3.680$, $p < .001$, with a large effect size ($r = .61$).

5.2 | RQ 2: Per student, is feedback effective?

Analysing the total number of compilation errors on which feedback was provided (18 error messages), the Mann–Whitney *U* test found that the experimental group (Mdn = 20) made significantly fewer compilation errors per student than the control group (Mdn = 45.50), $U = 1889$, $z = -3.99$, $p < .001$, medium effect size ($r = .32$).

For each of the 18 error messages on which feedback was applied, the total number of compilation error messages per student was then analyzed. As it is shown in Table 5, the Mann–Whitney *U* test found that the experimental group made significantly fewer compilation errors per student than the control group for 8 of the 18 error messages: CE_1, CE_2, CE_4, CE_5, CE_6, CE_7, CE_10, and CE_16.

To further explore the previous point, the number of compilation errors per student made in each of the sessions was analyzed, both for the total number of errors in that session and for each of the 18 error messages studied using the Mann–Whitney *U* test.

In the case of the first session (D&C), no significant differences were found in the number of errors per student between the experimental group and the control group, neither for the total of the 18 error messages, nor for each of them.

In the case of the second session (Greedy), the experimental group made significantly fewer compilation errors per student (Mdn = 1) than the control group (Mdn = 10.50), $U = 1917$, $z = -3.99$, $p < .001$, medium effect size ($r = .32$). In the individual analysis, as it is shown in Table 6, the experimental group made significantly fewer compilation errors per student in 12 of the 18 errors: CE_1, CE_4, CE_5, CE_6, CE_7, CE_8, CE_10, CE_11, CE_12, CE_13, CE_14, and CE_16.

In the case of the third session (DP), the experimental group made significantly fewer compilation errors per student (Mdn = 0) than the control group (Mdn = 1), $U = 1878$, $z = -4.7$, $p < .001$, medium effect size ($r = .38$). In the individual analysis, the experimental group made significantly fewer compilation errors per student in 6 of the 18 errors (all Mdn = 0): CE_1 ($U = 2250$, $z = -3.77$, $p < .001$, medium effect size [$r = .30$]), CE_2 ($U = 2435$, $z = -2.91$, $p = .004$, small effect size [$r = .23$]), CE_3 ($U = 2660.50$, $z = -2.55$, $p = .011$, small effect size [$r = .20$]), CE_5 ($U = 2699$, $z = -2.19$, $p = .029$, small effect size [$r = .18$]), CE_8 ($U = 2616.50$, $z = -2.62$, $p = .009$, small effect size [$r = .21$]), CE_16 ($U = 2810.50$, $z = -2.25$, $p = .024$, small effect size [$r = .18$]).

For the fourth and final session (BT), the experimental group made significantly fewer compilation errors per student (Mdn = 0) than the control group (Mdn = 5.50), $U = 4812.50$, $z = -4.47$, $p < .001$, medium effect size ($r = .36$). In the individual

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 5 Mann–Whitney *U* test significant results obtained for the 18 error messages analyzed.

	U	*Z*	*p*	*r*
CE_1	2061.50	−3.38	.002	.27 (medium)
CE_2	2441	−2.03	.043	.16 (small)
CE_4	2168.50	−3.05	.002	.24 (small)
CE_5	2113	−3.30	.001	.026 (small)
CE_6	2169.50	−3.11	.002	.24 (small)
CE_7	2342	−2.66	.008	.21 (small)
CE_10	2408.50	−2.23	.026	.18 (small)
CE_16	2595	−2.02	.043	.16 (small)

TABLE 6 Mann–Whitney *U* test significant results obtained for the second session (Greedy).

	U	*Z*	*p*	*r*
CE_1	2029	−3.82	<.001	.31 (medium)
CE_4	2064.50	−4.05	<.001	.32 (medium)
CE_5	2021.50	−4.18	<.001	.34 (medium)
CE_6	2342	−3.98	<.001	.32 (medium)
CE_7	2564	−2.38	.017	.19 (small)
CE_8	2459	−2.66	.008	.21 (small)
CE_10	2575	−1.99	.048	.16 (small)
CE_11	2607	−2.18	.029	.18 (small)
CE_12	2503	−2.80	.005	.22 (small)
CE_13	2651.50	−2.32	.020	.19 (small)
CE_14	2581.50	−3.14	.002	.25 (small)
CE_16	2731.50	−2.18	.29	.18 (small)

analysis, the experimental group made significantly fewer compilation errors per student in 5 of the 18 errors: CE_1 ($Mdn_{exp} = 0$, $Mdn_{ctrl} = 1$, $U = 2159$, $z = -3.36$, $p < .001$, small effect size [$r = .27$]), CE_2 ($U = 2489.50$, $z = -2.\ 46$, $p = .014$, small effect size [$r = .20$]), CE_3 ($U = 2553.50$, $z = -2.36$, $p = .018$, small effect size [$r = .19$]), CE_6 ($U = 2481.50$, $z = -2.87$, $p = .004$, small effect size [$r = .23$]), CE_11 ($U = 2661$, $z = -2.06$, $p = .039$, small effect size [$r = .17$]).

The table below shows the effect sizes for each session, both for total errors and for each error, where "↑" is large effect, "↓" is small effect, "↔" is medium effect y "." means no significant differences (Table 7).

TABLE 7 Effect sizes on a per-session and per-error basis.

	Greedy	Dynamic Programming	Backtracking
Globally	↔	↔	↔
CE_1	↔	↔	↓
CE_2	.	↓	↓
CE_3	.	↓	↓
CE_4	↔	.	.
CE_5	↔	↓	.
CE_6	↔	.	↓
CE_7	↓	.	.
CE_8	↓	↓	.
CE_9	.	.	.
CE_10	↓	.	.
CE_11	↓	.	↓
CE_12	↓	.	.
CE_13	↓	.	.
CE_14	↓	.	.
CE_15	.	.	.
CE_16	↓	↓	.
CE_17	.	.	.
CE_18	.	.	.

6 | DISCUSSION

The aim of the present study was to determine whether the formative feedback provided by the lecturer using COLMENA dash-boards helps to reduce the number of compilation errors made by students in a face-to-face learning context. In the subsequent section, we will discuss each of the research questions formulated on the basis of the results obtained.

6.1 | RQ 1: Per message error, is feedback effective?

The quick answer to this question is simply that the formative feedback provided by the lecturer with the help of COLMENA is effective. We approached the study of this question from two perspectives. Firstly, in the total sum of errors committed, there is a significant reduction in the number of errors, close to 50%, with the experimental group having done the same work and with the same lecturer as the control group. Secondly, in

the case of the total sum of errors per session, the results obtained indicate a large reduction in three of the four sessions.

These results demonstrate that the approach followed by COLMENA—which is focused on the error-specific concept—is more suitable than the approaches followed by Jadud [16] and Luke [21]. Thus, error collection related with a specific session during which the lecturer gives specific contents on a specific moment gives rise to a more efficient formative feedback as its nature is more immediate than the further analysis the lecturer may carry out about errors in a final delivery. The immediate nature COLMENA has allows corrective feedback and, therefore, it helps to be positive. Besides, error reduction of about 50% in the experimental group, confirms the need of having tools such as COLMENA which fosters the lecturer's formative feedback if compared with other approaches based on enhanced error message [9], automatic error detection and solving [1], repairing of programs [35] which have proved not to be practical for automatic generation of feedback because feedback nature is minimal [10].

A more detailed analysis by sessions allows us to go deeper into this question. Therefore, in the first session (D&C), where the students implement the D&C algorithm, no significant differences were found. In this session, the lecturer makes the first contact with the students, and for students in both groups is the first session. The data indicate the large amount of errors that exist in both groups for that first session, where the students are acquainting themselves with the subject, the lecturer and the working environment. Besides, students need knowledge to implement the D&C algorithm, such as recursion, which is one of the most difficult concepts to teach [7, 13, 15, 31] and understand, hence the high number of errors in both groups. It is noteworthy that in the experimental group there are fewer errors than in the control group, which is indicative of the help offered by COLMENA to the instructor, although it is true that this help is not significant.

The absence of significant differences in this first session and the large number of errors is in the same direction as previous work in which it was concluded that students do not possess the necessary background knowledge [8]. The lack of this knowledge makes it difficult for students to make critical connections between the feedback and the work done [29]. This is precisely what may be happening comparatively between the D&C and BT sessions, explaining why in the former there is no significant reduction in compilation errors and in the latter there is. In the BT session, the students may have already assimilated the knowledge discussed in D&C and, therefore, were able to interpret and apply the feedback received more easily.

6.2 | RQ 2: Per student, is feedback effective?

In this research question, we analyzed whether the feedback helped to reduce the errors in compilation per student, both globally and for each session. Overall, in 8 of the 18 messages studied (60.37% of the total number of errors made), we found a significant reduction in the number of errors made by each student. Thus, the feedback provided to the experimental group seems to be more effective than that provided to the control group, especially in the most frequent error messages (CE_1–CE_7). An exception is the error message CE_3 (Figure 3) whose frequency represents 6.75% of the total errors made by the experimental group. This error message generally appears when students try to use a class that is not imported or not accessible from the code fragment in which it is referenced. While the examples provided by COLMENA (Figure 3) detail how to handle the error at the code level, it is likely that students are failing to import external libraries through the IDE. We consider this relevant because COLMENA does not provide information on how to work with the IDE, so the error CE_3 is a good example that indicates the need to add complementary explanations in the tools on how to use the IDE. These explanations would be part of the more elaborate feedback [19] that might be included in feedback tools. Nevertheless, at present, this more elaborated feedback is not considered as automatic generation because, as it was explained in Section 2, none of the ways mentioned to generate formative feedback has solved the problem of feedback being task-specific, even less, being corrective or positive.

Taking each session individually, feedback helped reduce the number of errors per student in the same sessions as the previous research question, thus repeating the effect of the D&C session, where feedback did not help reduce the number of errors per student. In the sessions where feedback was effective, the reduction in errors per student occurred unevenly depending on the error message. The first time the reduction for some of the most frequent error messages (CE_1, CE_4, CE_5, and CE_6) occurred, we observe that it is comparatively much larger in the first few sessions than in later sessions, moving from a medium effect size to a small effect size. This seems to indicate that the experimental group's feedback is especially effective in the first sessions, reducing its effectiveness in later sessions. The fact that these differences are becoming smaller and smaller may be due to the positive and corrective effect of the feedback. That is, after seeing the error and fixing it because of the feedback received, students were less likely to generate the same error in the future [8].

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Along the same lines as other authors [8] we observed the existence of errors associated with a lack of knowledge. Thus, in four errors, CE_9, CE_15, CE_17, and CE_18, there is no difference between the control group and the experimental group. We believe that there is no behavioral change in the students in these four cases for the two groups, because a prior knowledge background is needed, which, if missing, leads to the occurrence of these errors. For instance, the error message CE_18 (return type for the method is missing) in our view, is an error typical of a novice student who probably does not fully understand the functionality of such a method and what it returns as a result. Another example is error CE_9 (Syntax error on token [PLACEHOLDER], delete this token). The appearance of this error can be due to multiple causes, from the absence of the "=" operator in the assignment of variables to the misuse of tokens in logical operations. Indeed, the proper use of logical expressions in the Greedy, PD, and BT techniques allows the algorithms to work properly. Specifically, in BT they are necessary to explore the tree of solutions to a problem, while in Greedy and PD certain values are selected, specific to the algorithm, which allow it to function correctly. If the student does not understand these techniques, it is very difficult for them to make proper use of logical expressions. This information confirms the need for immediate corrective feedback given by the lecturer, thanks to COLMENA. Even in the case of feedback related to a specific session, it has a positive effect as it reduces the presence of such errors in the future.

These results are not obtained with the approach of enhanced error messages [3] that, on some occasions, are notoriously problematic, especially for novices [9]. They are neither obtained with tools for detecting errors and repairing programs that have proved to be not really useful for achieving their purpose [35]. It is even impossible to get them by implementing any technique that may allow the generation of personalized feedback automated for the student, either for showing hints, suggestions [26] or examples [8], and that uses large scale samples as it is not practical to do it at a large scale because, at the end, manual examination of error solution will be required [1].

7 | CONCLUSIONS

Feedback contributes to reducing errors, but for it to be effective it has to be error-specific, immediate, positive, and corrective. Currently, research is aimed at generating automatic feedback, which is far from proving its effectiveness because it does not achieve the aforementioned characteristics. In this situation, the lecturer still remains a fundamental agent in the teaching- learning process of programming. Nonetheless, despite their protagonism, they have limited resources to provide adequate feedback in educational scenarios with many students and little time.

In this paper we have shown that by supporting the instructor's formative feedback with the right tools, such feedback could be effective. Through a case study with novice students, we have reached the conclusion that specific-error, immediate, corrective, and positive feedback reduces compilation errors and helps students to make fewer errors. Consequently, it is necessary to support the lecturer with tools such as the one we developed in this work, which we call COLMENA, because the generation of automatic feedback is not sufficiently advanced to offer this feedback to the student without the intervention of the lecturer.

We have found through a comparative study involving two groups that the control group (the lecturer gives formative feedback to the students without any support) has more errors associated with the 18 error messages than the experimental group (the lecturer gives formative feedback to the students with COLMENA support). The previous result is complemented by other more exhaustive results that demonstrate the effectiveness of the lecturer's formative feedback thanks to COLMENA technology. Therefore, for each error message, the feedback is effective for several reasons. Firstly, the number of compilation errors is significantly lower in the experimental group. Secondly, in a more detailed study of the four sessions, we found that in three of them there are significantly fewer compilation errors in the group of students who receive the formative feedback from the lecturer with the help of COLMENA. Additionally, a per-student study demonstrates the effectiveness of the lecturer's COLMENA-supported feedback. Thus, students in the experimental group make fewer errors than students in the control group. For the 18 error messages on which feedback is applied, in 8 of them the number of errors is significantly lower thanks to the COLMENA support. A more detailed analysis of the 18 messages in each session allows us to conclude that, in three of the four sessions, the feedback with COLMENA was effective, with the reduction of errors per student being uneven depending on the error message. From a course-wide perspective, errors are reduced for each error message.

Nevertheless, in spite of all the efforts we make to provide feedback to students, there is still a gap that needs to be studied in depth. In this research we have shown that formative feedback helps to correct compilation errors, however, we have also mentioned the necessity of a more elaborated feedback in two parallel and complementary lines, that is, the feedback associated to the use of IDEs and the one associated to compilation errors related to the lack of specific knowledge of each session. It is this line of elaborated feedback on which we

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

are currently working and which we hope will soon produce results. Furthermore, more research is needed with a larger sample, in different contexts and cultures, with nonsynchronous courses, in online subjects or in longer courses.

ACKNOWLEDGMENTS

This work has been partially funded by the Spanish Department of Science, Innovation and Universities: Project RTI2018-099235-B-I00. The authors have also received funds from the University of Oviedo through its support to official research groups (GR-2011-0040).

DATA AVAILABILITY STATEMENT

The data set is available to the research community upon request.

ORCID

MPuerto Paule-Ruiz https://orcid.org/0000-0003-0286-5430

REFERENCES

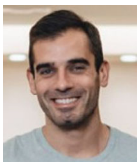
1. T. Ahmed, N. R. Ledesma, and P. Devanbu, *SynShine: Improved fixing of syntax errors*, IEEE Trans. Softw. Eng. **49** (2023), no. 4, 2169–2181.
2. B. A. Becker, *An effective approach to enhancing compiler error messages*, Proc. 47th ACM Tech. Symp. Comput. Sci. Educ., 2016, pp. 126–131.
3. B. A. Becker, P. Denny, R. Pettit, D. Bouchard, D. J. Bouvier, B. Harrington, A. Kamil, A. Karkare, C. McDonald, P.-M. Osera, J. L. Pearce, and J. Prather, *Compiler error messages considered unhelpful: The landscape of text-based programming error message research*, Proc. Work. Group Rep. Innov. Technol. Comput. Sci. Educ., 2019, pp. 177–210.
4. N. C. C. Brown, A. Altadmri, S. Sentance, and M. Kölling, *Blackbox, five years on: An evaluation of a large-scale programming data collection project*, Proc. 2018 ACM Conf. Int. Comput. Educ. Res., 2018, pp. 196–204.
5. J. Cohen, Statistical power analysis for the behavioral sciences, Routledge, New York, 1988.
6. J. Cohen, *A power primer*, Psychol. Bull. **112** (1992), no. 1, 155–159.
7. N. B. Dale, *Most difficult topics in CS1: Results of an online survey of educators*, ACM SIGCSE Bull. **38** (2006), no. 2, 49–53.
8. P. Dennya, A. Luxton-Reilly, and D. Carpenter, *Enhancing syntax error messages appears ineffectual*, Proc. 2014 Conf. Innov. Technol. Comput. Sci. Educ., 2014, pp. 273–278
9. P. Denny, J. Prather, B. A. Becker, C. Mooney, J. Homer, Z. C. Albrecht, and G. B. Powell, *On designing programming error messages for novices: Readability and its constituent factors*, Proc. 2021 CHI Conf. Hum. Factors Comput. Syst., 2021, pp. 1–15.
10. P. Denny, J. Whalley, and J. Leinonen, *Promoting early engagement with programming assignments using scheduled automated feedback*, Proc. 23rd Australas. Comput. Educ. Conf., 2021, pp. 88–95.
11. E. Duval, *Attention please! Learning analytics for visualization and recommendation*, 2011, pp. 9–17.
12. F. Gobet and G. Clarkson, *Chunks in expert memory: Evidence for the magical number four ... or is it two?* Memory **12** (2004), no. 6, 732–747.
13. K. Goldman, P. Gross, L. H. HeerenCinda, C. L. KaczmarczykLisa, and C. Zilles, *Setting the scope of concept inventories for introductory computing subjects*, ACM Trans. Comput. Educ. TOCE **10** (2010), no. 2, 1–29.
14. B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer, *What would other programmers do: Suggesting solutions to error messages*, Proc. SIGCHI Conf. Hum. Factors Comput. Syst., 2010, pp. 1019–1028.
15. M. Hertz, and S. M. Ford, *Investigating factors of student learning in introductory courses*, Proc. 44th ACM Tech. Symp. Comput. Sci. Educ., 2013, pp. 195–200.
16. M. C. Jadud, *An exploration of novice compilation behaviour in BlueJ*, Ph.D. thesis, University of Kent, 2006.
17. I. Karvelas, and B. A. Becker, *Sympathy for the (Novice) developer: Programming activity when compilation mechanism varies*, Proc. 53rd ACM Tech. Symp. Comput. Sci. Educ. **1** (2022), no. 1, 962–968.
18. A. M. Kazerouni, *Measuring the software development process to enable formative feedback*, Dissertation, Faculty of the Virginia Polytechnic Institute and State University, 2020.
19. F. van der Kleij, R. Feskens, and T. Eggen, *Effects of feedback in a computer-based learning environment on students' learning outcomes: A meta-analysis*, Rev. Educ. Res. **85** (2015), no. 4, 475–511.
20. J. Leinonen, P. Denny, and J. Whalley, *A comparison of immediate and scheduled feedback in introductory programming projects*, Proc. 53rd ACM Tech. Symp. Comput. Sci. Educ Vol. **1** (2022), no. 1, 885–891.
21. J. A. Luke, *Continuously collecting software development event data as students program*. 2015.
22. G. Marceau, K. Fisler, and S. Krishnamurthi, *Mind your language: On novices' interactions with error messages*, Proc. 10th SIGPLAN Symp. New Ideas New Paradig. Reflect. Program. Softw., 2011, pp. 03–18.
23. D. McCall, *Novice programmer errors—Analysis and diagnostics*, PhD thesis, University of Kent, 2016.
24. S. Parihar, Z. Dadachanji, P. K. Singh, R. Das, A. Karkare, and A. Bhattacharya, *Automatic grading and feedback using program repair for introductory programming courses*, Proc. 2017 ACM Conf. Innov. Technol. Comput. Sci. Educ., 2017, pp. 92–97.
25. R. S. Pettit, J. Homer, and R. Gee, *Do enhanced compiler error messages help students? results inconclusive*, Proc. 2017 ACM SIGCSE Tech. Symp. Comput. Sci. Educ., 2017, pp. 465–470.
26. P. M. Phothilimthana, and S. Sridhara, *High-coverage hint generation for massive courses: Do automated hints help CS1 students*, Proc. 2017 ACM Conf. Innov. Technol. Comput. Sci. Educ., 2017, pp. 182–187.
27. J. Prather, R. Pettit, K. H. McMurry, A. Peters, J. Homer, N. Simone, and M. Cohen, *On novices' interaction with compiler error messages: A human factors approach*, Proc. 2017 ACM Conf. Int. Comput. Educ. Res., 2017, pp. 74–82.
28. R. Rosenthal, The handbook of research synthesis, Russell Sage Foundation, 1994, pp. 231–244.

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

29. D. R. Sadler, *Beyond feedback: Developing student capability in complex appraisal*, Assess. Evaluat. High. Educ. **35** (2010), no. 5, 535–550.
30. M. C. Scheeler, K. L. Ruhl, and J. K. McAfee, *Providing performance feedback to teachers: A review*, Teach. Educ. Spec. Educ. **27** (2004), no. 4, 396–407.
31. A. E. Tew, and M. Guzdial, *The FCS1: A language independent assessment of CS1 knowledge*, Proc. 42nd ACM Tech. Symp. Comput. Sci. Educ., 2011, pp. 111–116.
32. E. Thiselton, and C. Treude, *Enhancing Python compiler error messages via stack overflow*, 2019.
33. M. Thurlings, M. Vermeulen, T. Bastiaens, and S. Stijnen, *Understanding feedback: A learning theory perspective*, Educ. Res. Rev. **9** (2013), 1–15.
34. V. J. Traver, *On compiler error messages: What they say and what they mean*. Adv. Hum. Comput. Interact. **2010** (2010), no. 3, 1–10.
35. K. Wang, R. Singh, and Z. Su, *Search, align, and repair: Data-driven feedback generation for introductory programming exercises*, Proc. 39th ACM SIGPLAN Conf. Program. Lang. Des. Implement., 2018, pp. 481–495.
36. C. Watson, F. Li, and J. Godwin, *BlueFix: Using crowd-sourced feedback to support programming students in error diagnosis and repair*, Advances in web-based learning (E. Popescu, Q. Li, R. Klamma, H. Leung, and M. Specht, eds.), vol. **7558**, Springer, Berlin, Heidelberg, 2012, pp. 228–239.
37. C. Watson, F. W. B. Li, and J. L. Godwin, *No tests required: Comparing traditional and dynamic predictors of programming success*, Proc. 45th ACM Tech. Symp. Comput. Sci. Educ., 2014, pp. 469–474.

AUTHOR BIOGRAPHIES

Miguel Sanchez-Santillan is a Lecturer in Department of Computer Science at the University of Oviedo. He received his M.Sc. degree in Computer Science and his Ph.D. from the University of Oviedo in 2017. He has participated actively in several national projects related with e-learning systems and adaptive systems. His research lines are addressed to learning analytics and e-learning systems. He is the author of several publications of *Journal Citation Report* (JCR).



Carlos Fernandez-Medina was born in Oviedo, Spain, in 1988. He received the B.S. degree in computer science from the University of Oviedo, Oviedo, Spain, in 2009, the M.S. degree in web engineering from the University of Oviedo, Oviedo, Spain, in 2011. Currently, he is Teaching Fellow of computer science at the University of Oviedo, Oviedo, Spain. His research focuses on learning technologies and web engineering.

Juan R. Perez-Perez is a Lecturer in Department of Computer Science at the University of Oviedo. He received his M.Sc. degree in Computer Science in 1996 and his Ph.D. from the University of Oviedo in 2006. From 1995 to 1999 he worked for an information technologies company in the research and development departments, building specific development environments and database connectors. He has participated actively in several national projects related with e-learning systems and adaptive systems. His Ph.D. topic is about collaborative development environment on the web.

MPuerto Paule-Ruiz is Associate Professor for the Department of Computer Science at the University of Oviedo. She received her M.Sc. degree in Computer Science in 1997 and her Ph.D. from the University of Oviedo in 2003. From 1997 to 2000 she worked for several international companies as software analyst. She has participated actively in several regional and national projects related with adaptive systems and e-learning systems. Her research lines are addressed to mobile Learning, learning analytics and e-learning systems. She is the author of several publications of *Journal Citation Report* (JCR).

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: M. Sanchez-Santillan, C. Fernandez-Medina, J. R. Perez-Perez, and M. P. Paule-Ruiz, *An empirical evaluation of the formative feedback supported by dashboard in the context of compilation error*, Comput. Appl. Eng. Educ. 2023;**31**:1289–1305.
https://doi.org/10.1002/cae.22640

10990542, 2023, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/cae.22640 by Readcube (Labtiva Inc.), Wiley Online Library on [06/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License