UNCODE: INTERACTIVE SYSTEM FOR LEARNING AND AUTOMATIC EVALUATION OF COMPUTER PROGRAMMING SKILLS

Conference Paper · July 2018

DOI: 10.21125/edulearn.2018.1632

3 authors, including:

J.J. Ramirez-Echeverry
National University of Colombia
11 PUBLICATIONS **29** CITATIONS
SEE PROFILE

Felipe Restrepo-Calle
National University of Colombia
50 PUBLICATIONS **232** CITATIONS
SEE PROFILE

UNCODE: INTERACTIVE SYSTEM FOR LEARNING AND AUTOMATIC EVALUATION OF COMPUTER PROGRAMMING SKILLS

F. Restrepo-Calle, J.J. Ramírez Echeverry, F.A. González

Universidad Nacional de Colombia (COLOMBIA)

Abstract

This paper presents an educational environment to learn how to program computers which integrates an automatic grading tool: UNCode. The educational environment offers summative and formative feedback, by evaluating the functionality of computer programs. Given a programming problem, students must write a solution and validate it through the automatic grading tool of the educational environment. Summative feedback is offered through verdicts of the programs, indicating whether the syntax, semantics and efficiency of the program are correct or present some type of error. Based on these verdicts, a grade is assigned to the solutions proposed by the students. Moreover, the environment also offers formative feedback through different mechanisms that support the student in the improvement of the proposed programming solutions, before these are submitted for assessment. These mechanisms include: students' learning tools, tools for feedback, and monitoring tools.

Keywords: computer programming, educational environment, automatic grading tool.

1 INTRODUCTION

One of the most demanded skills nowadays for engineering students is computer programming. They should be able to [1]: analyze and understand how programs work; master a programming language; develop computer programs (design, implement, test and debug); among others. However, learning these skills is not an easy task. Students require not only to solve problems using the concepts learned in class [2], but also they require to study and practice out of class [3]. Moreover, it has been demonstrated that the continuous feedback by the teachers makes the student's extra-class practical work more effective for their learning process. In this way, students can know what they could have done better or how they could correct their programs [4]. Nevertheless, evaluating programming assignments manually in an objective manner and at an appropriate time is particularly difficult [5-6]. The manual assessment of programming exercises is a time-consuming task. Each program must be tested and its source code analyzed with respect to various aspects such as syntax, semantics, efficiency and maintainability [2]. While the syntax can be verified automatically by a compiler/interpreter, the evaluation of the other three aspects must be carried out manually. This makes the evaluation process tiresome and prone to faults. In this way, the grade assigned to an exercise is affected to a large extent by factors such as tiredness, favoritism, or inconsistencies of the person who evaluates [7].

ICT support tools have been developed as alternatives to mitigate the above problems. Keuning, et al. [8] classify these tools into two categories, i.e.: automated assessment (AA) tools [9-15] and learning environments for programming [16-19].

Automatic assessment tools for computer programming assignments are also called automatic grading tools or, in many cases, simply online judges [9-10]. They evaluate programs considering a wide spectrum of characteristics, ranging from features that can be analyzed statically in the source code, such as: style, design and metrics; to aspects that are dynamically analyzed, such as program behavior given a set of specific inputs and expected outputs, efficiency, etc. [7]. At the same time, this type of tools can be classified into two main categories: automatic evaluation systems for programming competitions and automatic evaluation systems for computer programming education [10]. Traditionally, automatic grading tools have been used in programming competitions, such as: ACM/ICPC (ACM International Collegiate Programming Contest)[1] and the Google Code Jam[2], among others. Recently, they have also been used by commercial sites for competitive programming and

[1] https://icpc.baylor.edu/

[2] https://code.google.com/codejam/

ISBN: 978-84-09-02709-5

recruitment of programmers such as TopCoder[3] and HackerEarth[4]. On the educational field, these tools have been adapted from their competitive nature to support the automatic assessment of computer programming tasks developed by students; for example, in Mooshak [12], Fernandez Aleman, 2011 [20], Luo et al., 2008 [21], and Wang et al., 2016 [22]. In these academic scenarios, summative feedback is offered to the students through the verdicts of the programs, indicating whether the syntax, semantics and efficiency of the program are correct or present some type of error. Based on these verdicts, a grade is assigned to the solutions proposed by the students. Although experimental results have shown that these tools work effectively to improve the practical skills of students compared to traditional teaching methods [22] and the competitive aspect has proven to be an excellent motivation for students [23], these systems fall short as formative assessment tools due to the limited feedback they offer to the students.

On the other hand, the learning environments for programming brings together tools where students can perform various activities such as reviewing concepts, exploring examples and animations, simulating and debugging their programs, among others. This type of tools includes: simple programming environments, such as Logo and Scratch [16]; interactive environments, such as learning tools based on examples, visualizations and animations, or simulation environments [17]; and even intelligent tutoring systems supported by different artificial intelligence techniques [18-19]. Although these environments offer formative feedback through different mechanisms that support the students in their learning process before they are evaluated, generally these systems lack of grading tools, which could complement the feedback while the students practice.

In this context, we present UNCode, an educational environment for learning computer programming skills and the automatic assessment of computer assignments in an academic context, which combine the best of both worlds, automatic grading tools and learning environments. UNCode offers, at the same time, summative and formative feedback to the students, which can contribute significantly to support difficulties associated to learning computer programming and evaluating programming tasks.

The rest of this paper is organized as follows. Section 2 tells about our previous experiences using automating grading tools in computer programming courses and the reasons why we considered necessary to develop another tool. Section 3 presents UNCode, our proposed educational environment for computer programming. Section 4 compares UNCode to other systems qualitatively. Finally, Section 5 concludes this paper and devises lines for the future work.

2 PREVIOUS EXPERIENCE

Since 2013, automatic grading systems have been used for the assessment of several computer programming courses at the Universidad Nacional de Colombia, such as Data Structures and Algorithms. The methodologies of these courses require a considerable amount of practical work, developing computer programs to solve the proposed problems. Automatic grading systems help to offer timely feedback about the results of the programs developed by each of the students. For this purpose, we selected the online judge called DomJudge as a support tool for the automatic grading of computer programs [24]. In this way, the evaluation is based mainly on assessing students' computer programming skills continuously and frequently.

DomJudge [24] is an online judge that allows students to submit their solutions (source code of computer programs) to the proposed programming problems, obtaining an immediate feedback. This is possible by evaluating different test cases where the program inputs and expected outputs are previously known, as well as additional restrictions, such as: the ranges of values between which the program inputs will be given, and considerations for estimating the maximum execution time of the programs.

Due to DomJudge was a tool oriented mainly to support programming competitions, it was necessary to adapt the way of using the assessment tool to support the needs of an academic environment. For example, a set of problems (a contest in the DomJudge) was set up for each of the practical projects proposed in the course. The students submitted their programs during the 2 hours that the evaluation lasted. For each problem, the students sent their source code in the Java, C/C++ or Python languages to the platform to be judged. The judge performs the automatic assessment of the programs in terms

[3] https://www.topcoder.com/

[4] https://www.hackerearth.com/

of syntactic correction, semantic correctness and efficiency. This facilitates the student to identify almost immediately if their program solves the problem effectively or what kind of aspects should be reviewed. If the program does not meet these requirements, the student has two hours to correct it. In case the students decide to make corrections to their programs, they could make multiple attempts that will be evaluated and stored in the judge without any penalty.

Each of the programs submitted to the judge's evaluation receives a verdict in accordance with its results. These verdicts serve as feedback to students and correspond to the following:

- Correct: the program is correct at the lexical, syntactic and semantic level; and meets the minimum required efficiency requirements.
- Compiler-error: the solution is not correct at the lexical or syntactic level.
- Wrong-answer: the program is not correct at the semantic level, i.e., for some test cases of inputs, the program obtained different outputs than the expected ones.
- Timelimit: the program exceeded the maximum allowed execution time, i.e., it is not correct in terms of efficiency.
- Run-error: an exception occurred at runtime and the program did not finish its execution correctly. For example: division by zero, access to an invalid memory location, etc. This error can be interpreted as a problem in the implementation at the semantic level.
- No-output: the execution of the program did not produce any output and it was not possible to evaluate it. This error can also be interpreted as a semantic problem.

Notice that the only desired verdict is ‘correct’, others indicate different errors within the program.

This tool not only facilitated to the teacher the assessment of large amounts of programs, but also provided immediate summative feedback to the students. In that sense, the tool was used to assign a grade to the solutions proposed by the students. Other advantages of adopting this kind of tools in computer programming courses have been widely reported in the literature [7]. However, the provided feedback of this tool has a coarse granularity level; for example, the judge informs to a student that her program is obtaining an incorrect answer, or that the program is not efficient enough in terms of the execution time, but the judge does not give her more details about where he could have the problem or how to correct it.

Moreover, from this previous experiences, we have studied several aspects to understand particularities related to how students learn to program computers. For instance, relationships between self-regulation in learning and students' source code style [25]; relationships between the student's code style and their personality [26]; and characteristics of self-regulation in the learning of students in a programming course and its relationship with the academic performance of students [27]. From the latter, in particular, we found that that self-regulated learning variables such as anxiety and self-efficacy for learning have a strong influence in the students’ academic performance [27]. This might be a consequence of the inference of the evaluation method based on an automatic grading tool (DomJudge).

3 UNCODE

Besides the summative feedback offered to the students by an automatic assessment tool, it is necessary to provide formative feedback to the students during the learning process. Therefore, we propose UNCode, which is a web-based educational environment for learning computer programming skills and the automatic assessment of computer assignments in an academic context. It offers summative and formative feedback to the students by means of a software tool that combines an automatic grading tool with several functionalities of learning environments. UNCode is deployed at https://www.ingenieria.bogota.unal.edu.co/uncode. Its source code is publicly available at https://github.com/JuezUN and it is distributed under AGPL 3.0 license[5].

Similarly to other automated assessment tools in academic scenarios, given a programming assignment, students must write a program (source code) of a possible solution and validate it through the grading tool (also known as online judge). UNCode is built on top of INGInious[6], which is a secure

[5] GNU Affero General Public License (https://www.gnu.org/licenses/agpl-3.0.en.html).

[6] https://www.inginious.org/

and automated code assessment platform developed in the Catholic University of Louvain. In this way, summative and immediate feedback is offered to the students by means of the verdicts of the submissions (solution attempts), indicating whether the syntax, semantics and efficiency of the program are correct or have some kind of problem. Based on these verdicts, a grade is assigned to the solutions proposed by the students. Moreover, unlike most automated assessment tools in the literature, UNCode also provides formative feedback through different tools that support the student in the improvement of the proposed solutions, even before the solutions are submitted for evaluation. These tools include: students' learning tools, monitoring tools, and especial functionalities designed to facilitate to provide a more detailed feedback. Fig. 1 presents the general architecture of UNCode.

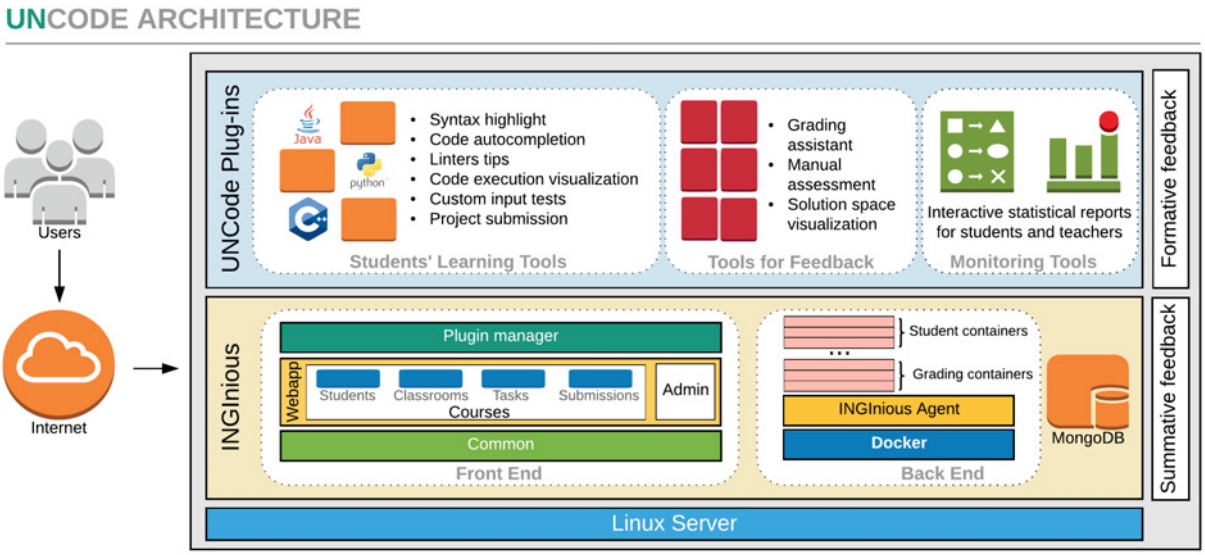

Figure 1. UNCode: educational environment for learning computer programming skills and the automatic assessment of computer assignments in an academic context.

3.1 INGInious

INGInious [28-29] is an automatic grading system for programming exercises. From a technical point of view, it offers a flexible, secure, scalable and extensible architecture, which was convenient to build UNCode on top of it. In addition, it is conceived as a tool to support educational environments, instead of competitive programming contests. INGInious is also a free and open-source software distributed under the AGPL license, and its sources are available on GitHub [28]. Other tools worth to explore to this purpose are: Autolab[7] (educational) and CMS - Contest Management System[8] (competitive).

This tool is comprised of two main components, namely the front-end and the back-end. The frontend provides a web interface (webapp) for the students and teachers, which depending on the user role permits to manage courses, students, classrooms, tasks and submissions; it also has an administration tool to configure the application; and it offers a plugin manager, which facilitates the extensibility of the system. Moreover, the back-end main responsibilities are: to manage the database (MongoDB); to create, manage and delete Docker[9] containers (to securely isolate students code execution and grading in a lightweight way); to run and grade the code made by the students.

The main features of INGInious can be summarized as follows: web-based interface; secure: students' code is executed in sandbox using Docker containers; customizable program limits: configurable execution time and memory limits for students' code; scalable: front-end and back-end can scale horizontally; extensible by means of plugins; support for LMS (Learning Management Systems) like Moodle[10]; free and open-source.

[7] http://www.autolabproject.com/

[8] https://cms-dev.github.io/

[9] https://www.docker.com/

[10] http://moodle.org/

3.2 UNCode Plug-ins

A set of plugins was developed to provide formative feedback to the students. These tools are aimed at clarifying to the students about the level of understanding that they have obtained of the subjects and of how they are performing the tasks. That is, the students receive corrective information about what they need to understand, allow them to deduce what learning strategies to use to facilitate the completion of the task, and can help them to find reasons that motivate them to increase their effort and commitment in the work of learning [30]. As can be seen in Fig. 1, the UNCode plug-ins can be grouped in three categories: students' learning tools, tools for feedback, and monitoring tools.

3.2.1 Students' learning tools

These tools are aimed at supporting the learning process of the students during the development of the solution of the programming assignments. That is, students do not have to wait until they make a submission of a solution in order to obtain feedback, they can get formative feedback along the process in several ways, as can be seen next. These tools are supported for Java, Python, and C/C++. The developed tools include the following:

Syntax highlight: we have integrated a JavaScript component called CodeMirror[11], which is a rich code editor for UNCode which runs in the web browser. CodeMirror main functionalities include: syntax highlighting for several programming languages, auto indentation and outdent, and the possibility to handle large documents without trouble. This means that the students can develop their programs using the editor directly on the web, which gives them the coding user experience of traditional IDEs (Integrated Development Environments). Formative feedback is provided to the students on-the-fly because syntax errors will be highlighted on the editor immediately while they are writing the code. In this way, students do not have to wait until they make a submission of the program to realize they have made a mistake related to the syntax of the language.

Code autocompletion: using a CodeMirror's add-on, UNCode code editor shows autocompletion hints. Depending on the code being written, and its available objects, it pops up a widget that allows the student to select among a set of completion options. This feature can be considered as formative feedback due to it helps students to familiarize with the syntax and semantics of the language.

Linters: we have also integrated linters to the code editor in UNCode for several programming languages. A linter is a software tool that analyze source code to flag programming errors, bugs, stylistic errors, and suspicious constructs [31]. For flexibility, we have integrated Coala[12], which is a unified interface for linting and fixing code, regardless of the programming languages used. Coala has a set of plugins for several languages. In particular we are using linters for popular languages such as C/C++ (OClint), Python (Pylint) and Java (JavaPMD). Linters are useful to recommend to the students the best programming practices while they are learning. These tools help students to deal with code maintainability issues in advance. It is worth noting that in the majority of the cases these issues are not addressed when traditional automatic grading systems are used, even though code maintainability should be a required skill for good programmers. For that reason, UNCode provides formative feedback on code maintainability to the students.

Code execution visualization: in the same manner, we have integrated a visualization tool into UNCode to see a graphical representation of the code execution, i.e., Python Tutor[13]. Using this tool, teachers and students can write programs directly in the web browser, step forwards and backwards through the program execution to view the run-time state of data structures and variables [32]. This helps students to learn computer programming allowing them to understand what happens as the computer runs each line of code in their programs. The students can explore the visualizations interactively, at their own pace. In this way, the students receive information about what they need to understand, clarifying how they are performing the assignments.

Custom input tests: considering that the grading tool evaluates the programs using test cases, comparing the functionality of students' code (outputs) against a set of expected outputs, UNCode offers to the students the possibility to test their programs using their custom inputs. The system executes the programs giving these inputs and show the obtained results to the students. These tests

[11] https://codemirror.net/

[12] https://coala.io/

[13] http://pythontutor.com/

are not considered for grading, instead they become a powerful tool for practicing and testing. In this way, students can test the program inside the learning environment before submitting it for evaluation. Using this feature, students are able to test a wide spectrum of input cases ranging from basic input examples to complex corner cases.

Project submission: the majority of automatic grading tools only support code submissions made of a single source code file (or a single text area in the user interface). However, for most students' assignments it is necessary to consider multiple files submissions, i.e., complete software projects including several source code files, libraries, and other dependencies. For instance, in Java, a common use case can be seen in Object-Oriented Programming when a typical class diagram from UML is implemented in several code files. Therefore, UNCode supports this kind of multiple code files submissions, sending a compressed file with all the necessary files. This feature is supported for Java, Python, and C/C++:

- Java: submissions for projects written in Java are supported for 1.7 and 1.8 JDK versions. To submit a Java project a zip file containing a "src" folder and a "lib" folder must be provided. The "src" is the folder where all the code written must be located, this includes a Main.java file which will start the program execution. The "lib" is the folder where all the dependencies used must be located, this includes .jar files and .class files.
- C/C++: the compilers GCC and G++ 4.8.5 are installed. To submit a C/C++ project a zip file must be provided. This zip file must contain the source code and dependencies in the way the user prefers, thus, a Makefile has to be submitted within the zip file. The makefile is responsible for compiling the project and execute it.
- Python: a project can be submitted either in Python 2 or Python 3. The only requirement is to have a main.py script which would be the entry point for the application. Many folders and subfolders as needed can be included. Note, however, that the main.py script must be put in the root of the compressed file.

3.2.2 Tools for feedback

These tools are aimed at facilitating that teachers provide meaningful feedback to their students. We achieve this objective by means of three components:

Grading assistant: this tool is used by teachers to interactively generate the grader of each one of the assignments. This permits to configure the way that INGInious will grade the students' code. In this way, it is possible to upload test cases for the assignments and configure, if necessary, partial grading for the tests. In addition, teachers can select which test cases are shown to the students in case a different output is obtained. This helps students to identify what they could have done in a better way or which input cases should consider for their programs. In some cases, this permits to identify that the design of the whole programs should be reconsidered for efficiency purposes.

Manual assessment: considering that teaching assistants can provide extra feedback to the students by manually assessing a submission, or annotating the automatic grades assigned by the judge, this is an UNCode module that permits combining both manual and automated assessment. The teacher can view the student submissions, compare them to reference solutions, and complement the automated assessment by setting manual grades and including a comment in the submission for the students' feedback (see Fig. 2).

Solution space visualization: this is an interactive tool for analysis of a set of source code submissions made by students when solving a programming assignment. The goal of the tool is to give a concise but informative overview of the different solutions submitted by the students, identifying groups of similar solutions and visualizing their relationships in a graph. This tool helps teachers to evaluate students, by means of the visual representation of the set of submissions of a programming assignment. In particular, the tool compares the source code of the different submissions and graphically displays the similarities and dissimilarities as well as the clusters that group similar solutions. In this way, taking into account the verdicts obtained by the solutions, it provides information that may help the instructor to understand how students address the different programming problems, the kind of solutions that they submit, and the mistakes they make. This facilitates to provide more custom feedback to the students. This tool was previously published in [33] and we have integrated it to UNCode. However, it is worth mentioning that this module only supports Java code currently.

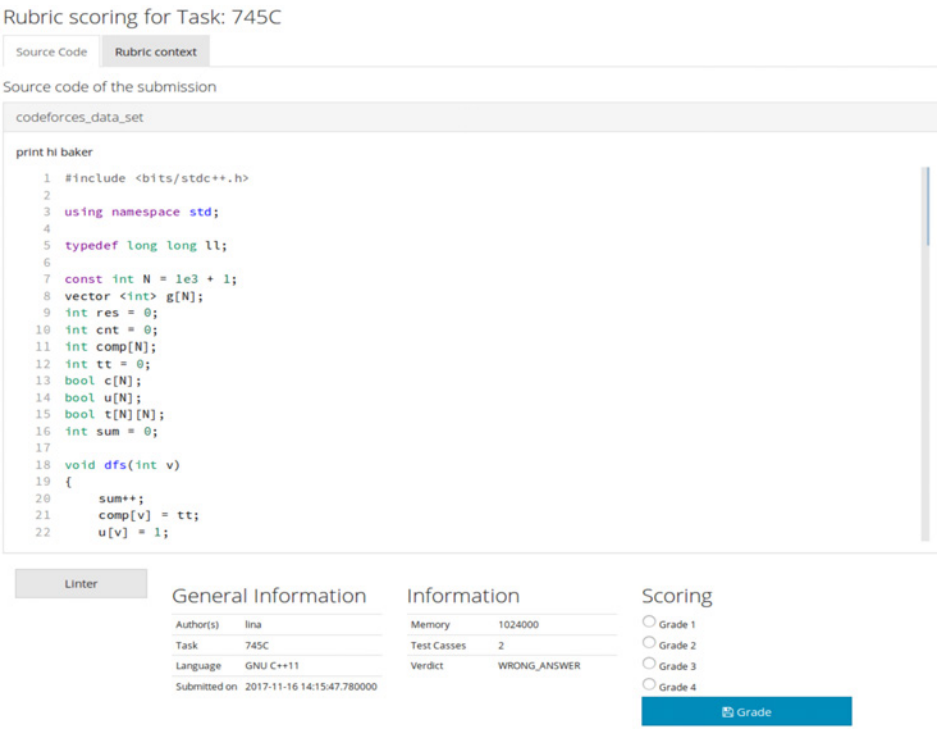

Figure 2. Screenshot of the UNCode's manual assessment module.

3.2.3 Monitoring tools

Aware of the importance of monitoring in self-regulated learning [34], UNCode provides interactive statistical reports for both students and teachers. Fig. 3 presents two screenshots of reports that every student can visualize; in this way, each one of the students can monitorize their own progress during the course. On the left side of Fig. 3, a bubble chart is presented showing the best grade obtained in each one of the tasks, and the size of the bubbles represents the number of submissions (attempts - trials) that the student did to achieve that grade. On the right side of Fig. 3, a stacked bar chart is shown with the distribution of the verdicts given to the submissions for every tasks. These charts are interactive as they are built using Plot.ly[14]. This data can be downloaded to CSV (Comma Separated Values) files.

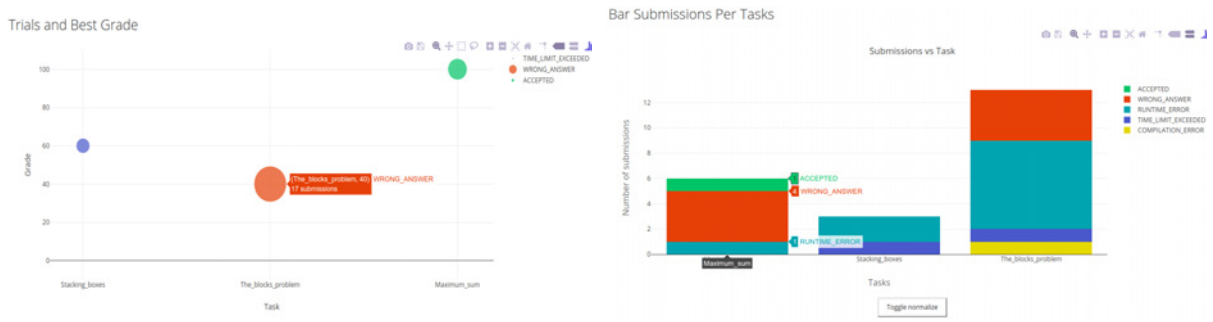

Figure 3. Screenshots of the students' interactive statistical reports.

Fig. 4 shows two screenshots of reports that teachers can revise. On the left side of Fig. 4, a bubble chart is presented, illustrating the students' grades for each assignment; the bubble size represents the number of submissions that all the students did to achieve that grade. On the right side of Fig. 4, a stacked bar chart is shown with the distribution of the best verdicts achieved by the students for every assignment. The most important thing to consider is that these charts are interactive, and teachers

[14] https://plot.ly/

can select bubbles or column areas to obtain a list of submissions of special interest. This permits to customize the feedback to the students.

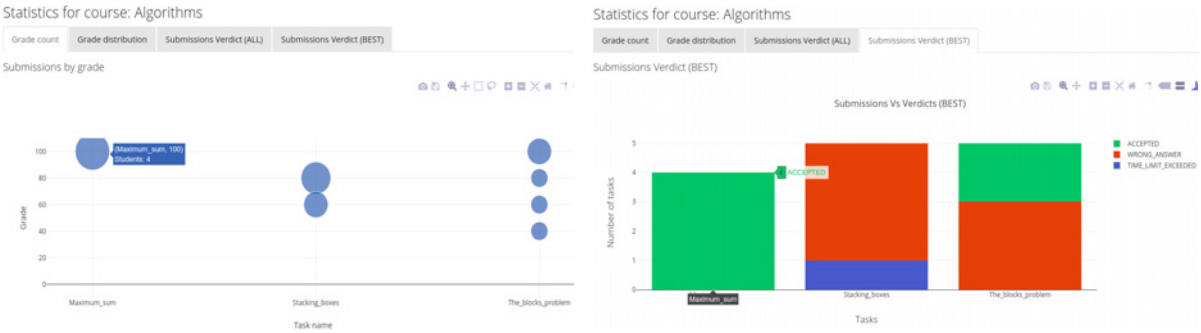

Figure 4. Screenshots of the teachers' interactive statistical reports.

4 RESULTS: QUALITATIVE COMPARISON TO OTHER TOOLS

Although UNCode is a work in progress tool and has not been tested in an academic environment yet, we believe that its new features for formative feedback will encourage active engagement of students when learning computer programming, rather than a passive attitude. So far, we have compared UNCode, in a qualitative way, to DomJudge (the previous automating grading tool) and INGInious. Table 1 presents this comparison.

Table 1. *Comparative between DomJudge, INGInious, and UNCode.*

Type	Feature	DomJudge	INGInious	UNCode
Summative feedback	Syntax			
	Semantics (functionality)			
	Efficiency			
	Numerical grading			
	Resubmissions			
	Partial grading		*	
	Automatic grading (numerical)			
	Manual grading			
Formative feedback	Syntax highlight		*	
	Code autocompletion		*	
	Code maintainability (linters)			
	Code execution visualization			
	Custom input tests			
	Project submissions			
	Highly customizable grading assistant			
	Solution space visualization			
	Scoreboard			
	Basic statistical reports			
	Interactive statistical reports			
Technical features	Programming languages	Many	Many	Java, Python, C/C++
	Web-based			
	User administration			
	Courses and assignments manager			
	Sandboxing			
	Scalability			
	Support for LMS (Learning Management Systems)			
	Distribution and Availability			

* Configurable

5 CONCLUSIONS

We have presented UNCode: a free, open-source, web-based educational environment for learning computer programming skills and the automatic assessment of computer assignments. This tool leverages the best of automatic assessment tools, and integrates learning environment features to provide, at the same time, summative and formative feedback to the students. UNCode can contribute to support difficulties associated to computer programming education and the assessment of programming assignments. Although UNCode's technical tests have been carried out successfully, it is necessary to validate the impact of each one of the proposed modules in an academic environment, using UNCode as a support system in several programming courses. In our future work, we will study this impact taking into account mainly the assessment given by students based on their experiences when using this tool in in the programming assignments and practices.

ACKNOWLEDGEMENTS

This project has been funded by the Universidad Nacional de Colombia (HERMES: 35713). We would also like to thank to the developers for their contributions: Daniel Castro Alvarado, Andrés Mauricio Rondón Patiño, Nicolás Larrañaga Cifuentes, Maikol Bonilla Gil, Lina Rosales Castro, Milder Hernandez, Cristian González Carrillo, Jesus Chavarro Muñoz, and Juan David Valencia Angulo.

REFERENCES

[1] M. Sahami and S. Roach, "Computer science curricula 2013 released," Communications of the ACM, vol. 57, no. 6, pp. 5–5, Jun. 2014.

[2] J. Carter et al., "How shall we assess this?" ACM SIGCSE Bulletin, vol. 35, no. 4, p. 107, Dec. 2003.

[3] S. Alhazbi, "Using e-journaling to improve self-regulated learning in introductory computer programming course," 2014 IEEE Global Engineering Education Conference (EDUCON), pp. 352–356, Apr. 2014.

[4] Souza, D. M., Felizardo, K. R., & Barbosa, E. A Systematic Literature Review of Assessment Tools for Programming Assignments. In 2016 IEEE 29th Int Conf on Software Eng Education and Training (CSEET) (pp. 147–156). 2016. IEEE. http://doi.org/10.1109/CSEET.2016.48

[5] Fonte, D., Da Cruz, D., Gançarski, a L., & Henriques, P. R. A flexible dynamic system for automatic grading of programming exercises. 2nd Symp on Languages, Apps and Tech, SLATE 2013, 29, 129–144. 2013.

[6] Cheang, B., Kurnia, A., Lim, A., & Oon, W. C. On automated grading of programming assignments in an academic institution. Computers and Education, 41(2), 121–131. 2003.

[7] Ala-Mutka, K. M. A Survey of Automated Assessment Approaches for Programming Assignments. Computer Science Education, 15(2), 83–102. 2005.

[8] Keuning, H., Jeuring, J., & Heeren, B. Towards a Systematic Review of Automated Feedback Generation for Programming Exercises. In Proceedings of the 2016 ACM Conf on Innovation and Tech in Computer Science Education - ITiCSE '16 (pp. 41–46). New York, New York, USA: ACM Press. 2016.

[9] Kurnia, A., Lim, A., & Cheang, B. Online Judge. Computers & Education, 36(4), 299–315. 2001.

[10] Ihantola, P., et al. Review of recent systems for automatic assessment of programming assignments. In Proc of the 10th Koli Calling Int Conf on Computing Education Research Koli Calling 10 (pp. 86–93). 2010.

[11] Caiza, J.C. and Álamo Ramiro, J. M. Programming assignments automatic grading: review of tools and implementations. In: "7th Intl Technology, Education and Development Conference (INTED2013)", March 2013, Valencia, Spain. pp. 5691-5700. 2013.

[12] Guerreiro, P., et al. Combating anonymousness in populous CS1 and CS2 courses. In Proc 11th annual SIGCSE conf on Innovation and tech in computer science education - ITICSE '06 Vol. 38, p. 8. 2006.

[13] Derval, G., Gego, A., Reinbold, P., Frantzen, B., Roy, P. Van, & Louvain, U. De. Automatic grading of programming exercises in a MOOC using the INGInious platform, 86–91. 2015.

[14] Zen, K., et al.. Using latent semantic analysis for automated grading programming assignments. Int Conf on Semantic Technology and Information Retrieval, STAIR 2011, June 28-29, 2011, pp. 82–88. 2011.

[15] José Paulo Leal. Managing programming contests with Mooshak. Software—Practice & Experience, 2003.

[16] Guzdial, M. Programming Environments for Novices. Comp science education research. pp. 127–154. 2003.

[17] Gomez-Albarran, M. The Teaching and Learning of Programming: A Survey of Supporting Software Tools. The Computer Journal, 48(2), 130–144. 2005. http://doi.org/10.1093/comjnl/bxh080

[18] Le, N.-T., et al. A Review of AI-Supported Tutoring Approaches for Learning Programming. pp. 267-279. 2013. Springer, Heidelberg. http://doi.org/10.1007/978-3-319-00293-4_20

[19] John C Nesbit, et al. Work in Progress: Intelligent Tutoring Systems in Computer Science and Software Engineering Education. In 2015 ASEE Annual Conf & Exposition. Seattle, Washington. 2015.

[20] Fernandez Aleman, J. L. Automated Assessment in a Programming Tools Course. IEEE Tran on Education, 54(4), 576–581. 2011. http://doi.org/10.1109/TE.2010.2098442

[21] Luo, Y., et al. Programming grid: a computer-aided education system for programming courses based on online judge. Proc 1st ACM Summit on Comp Education in China SCE'08 p. 1. 2008.

[22] Wang, G. P., et al. OJPOT: online judge & practice oriented teaching idea in programming courses. European Journal of Engineering Education, 41(3), 304–319. 2016.

[23] Lehtonen, T. Javala - addictive e-learning of the Java programming language. Proc of the Koli Calling 2005 Conference on Computer Science Education, November 2005, Koli, Finland. 2005. p. 41-48. 2005.

[24] Eldering, J., et al.: DOMjudge - programming contest jury system. http://www.domjudge.org/

[25] H. Castellanos, F. Restrepo-Calle, F. A. González and J. J. Ramírez Echeverry. Understanding the relationships between self-regulated learning and students source code in a computer programming course. In Proc of the 2017 IEEE Frontiers in Education Conf. (FIE). Indianapolis, IN, 2017, pp. 1-9.

[26] Francisco Rangel, et al. PAN at FIRE: Overview of the PR-SOCO Track on Personality Recognition in SOurce COde. Working notes of FIRE. Dec 7-10. Calcuta, India. 2016.

[27] Jhon Jairo Ramírez Echeverry, Lina F. Rosales-Castro, Felipe Restrepo-Calle, Fabio A. González. "Self-regulated learning in a computer programming course", in IEEE-RITA Revista Iberoamericana de Tecnologías del Aprendizaje, Vol. 13 Issue 2. Pp. 1-9. In press. May 2018. DOI: 10.1109/RITA.2018.2831758.

[28] Derval, G., Gégo, A., & Reinbold, P. (2014). INGInious [software]. https://github.com/UCL-INGI/INGInious

[29] Derval, G., et al. Automatic grading of programming exercises in a MOOC using the INGInious platform, In the Proc of the European MOOC Stakeholder Summit - EMOOCs 2015. pp. 86–91. 2015.

[30] Hattie, J., & Timperley, H. "The power of feedback". Review of Educational Research, vol. 77(1), 81-112, 2007. http://dx.doi.org/10.3102/003465430298487

[31] Darwin, Ian F. Checking C Programs with Lint: C Programming Utility (Revised ed.). United States: O'Reilly Media. 1991. ISBN 978-0937175309.

[32] Philip J. Guo. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. SIGCSE 2013.

[33] Rosales-Castro L.F., Chaparro-Gutiérrez L.A., Cruz-Salinas A.F., Restrepo-Calle F., Camargo J., González F.A. An Interactive Tool to Support Student Assessment in Programming Assignments. In: Advances in Artificial Intelligence - IBERAMIA 2016. LNCS vol 10022. pp 404-414. Springer. 2016.

[34] Manso-Vazquez, M., & Llamas-Nistal, M. A Monitoring System to Ease Self-Regulated Learning Processes. IEEE Revista Iberoamericana de Tecnologías Del Aprendizaje, 10(2), 52–59. 2015.