ISSN 0798 1015

REVISTA ESPACIOS

HOME Revista ESPACIOS ÍNDICES / Index A LOS AUTORES / To the AUTORS

EDUCACIÓN • EDUCAÇÃO • EDUCATION Vol. 39 (Number 35) Year 2018. Page 11

Competitive programming and gamification as strategy to engage students in computer science courses

Programación competitiva y ludificación como estrategia para motivar a los estudiantes de cursos de ciencias de la computación

Julián MORENO Cadavid 1; Andrés F. PINEDA Corcho 2

Received: 04/04/2018 • Approved: 29/05/2018

Contents

ABSTRACT:

We present a model for designing and implementing computer programming courses based on two approaches: competitive programming and gamification. From the former, it considers the ACM-ICPC standard for the exercises, as well as an automatic code validator. From the later, it considers the MDA model using several elements like points, rankings, levels and badges. For its validation, we conducted an experiment whose results showed that students' perception about the combined strategy was positive and that their academic performance was improved.

Keywords: Computer programming, ACM-ICPC, MDA, Learning Management System.

RESUMEN:

Presentamos un modelo para el diseño e implementación de cursos de programación de computadores basado en dos aproximaciones: programación competitiva y ludificación. De la primera, se considera el estándar ACM-ICPC para los ejercicios, así como un validador de códigos automático. De la segunda, se considera el modelo MDA haciendo uso de varios elementos como puntos, posiciones, niveles y medallas. Para su validación, llevamos a cabo un experimento cuyos resultados mostraron que la percepción de los estudiantes sobre la estrategia combinada fue positiva, al tiempo que mejoró su rendimiento académico.

Palabras clave: Programación de computadores, ACM-ICPC, MDA, Sistema de Gestión de Conocimiento.

1. Introduction

Teaching computer programming is not an easy task (Bennedsen, 2008) and is considered one of the biggest challenges in computer sciences (McGettrick et al., 2005, Wegner et al., 1996). In fact, there is a lot of evidence indicating a poor performance of college students

after completing the first programming course (Kranch, 2012; McCracken et al., 2001) as well as high dropout rates (Robins, Rountree & Rountree, 2003).

Regarding this phenomenon, several authors have identified and classified the main issues faced by students in programming courses (Koffmann & Torsten, 2001; Milne & Rowe, 2002; Lahtinen, Ala-Mutka, & Järvinen, 2005; Mow, 2008; Renumol, Dharanipragada & Janakiram, 2010). Such issues may be summarized in four areas: 1) Teaching methods: teaching is not personalized due to large groups, there is few or no feedback at all from the teacher to students, dynamic concepts are taught through static materials, and teachers are usually more focused on teaching the details of language syntax rather than promoting problem-solving using the programming language; 2) Study methods: students use incorrect study methodologies, do not know how to solve problems, do not have the background needed, do not have previous knowledge or experience with programming, and do not practice enough to acquire programming skills; 3) The nature of computer programming itself: programming requires a high level of abstraction, and programming languages often have very complicated syntax; 4) Psychological and social effects: students are usually not motivated, feel anonymous during classes, and there is few or no interaction among them.

In order to face some of the issues listed above, different strategies have been studied, being two of them competitive programming and gamification. For the one hand, competitive programming may be defined as a methodology to create environments that allow students for practicing together and competing in a healthy manner by solving coding exercises under a certain standard, usually the ACM-ICPC (Revilla, Manzoor & Liu 2008). Other authors flesh out that the primary goal of this approach is to engage students to be excellent coders by using the state of the art in algorithms and data structures to find the best solutions for the problems they face (Halim & Halim, 2010). For the other hand, gamification may be defined as the use of game mechanics to appeal to a particular audience and promote desired behaviors (Li et al., 2013). Some authors also define gamification as the use of game elements in non-game environments to empower engagement and motivation (Deterding et al., 2011; Schoech et al., 2013).

Both strategies have been widely validated, although separately. For instance, Garcia-Mateos & Fernandez-Aleman (2009) present the experience they had with competitive programming at the University of Murcia in Spain and specifically with the use of an automatic codes judge. According to the authors, this methodology encourages independent work, promotes competition and makes courses more appealing for students. Halim & Halim (2010) present a similar experience but in the National University of Singapore. More precisely, they describe the use of a competitive programming course to train students to participate in a worldwide competition. They show how students, who are part of this type of competitions, usually have higher performances and obtain better scores in their graduation programs than those who are not. Coles, Jones, and Wynters (2011) describe a methodological guide for the analysis of problem-solving skills to participate in programming contests. Even if they use an automatic judge to validate the codes, as is usual in competitive programming, the guide focuses on how students should solve problems rather than obtaining the final output. Combéfis and Wautelet (2014) present a guide for using online programming contests to support the teaching of programming. Also, they present an online platform that allows people for creating competitor profiles and compare themselves to other users as well as discussing the contest they took part in. According to the authors, such a proposal aims at increasing the motivation of people when learning computer programming and at promoting computer science among the younger ones.

Leaving aside competitive programming, Schoech et al. (2013) present PeerSpace, a gamified environment that allows computer science students for participating in discussion forums about programming problems. According to the authors, the use of this environment promotes competition among students, which are evaluated by classmates. Ibanez, Di Serio, and Delgado-Kloos (2014) present a study about the effectiveness and commitment of students in a gamified programming course in language C. The authors demonstrated that such students obtained better results than those in regular courses and argued that this phenomenon is due to the student motivation to overcome the regular milestones of this kind of courses. Akpolat and Slany (2014) studied the use of gamification in a software

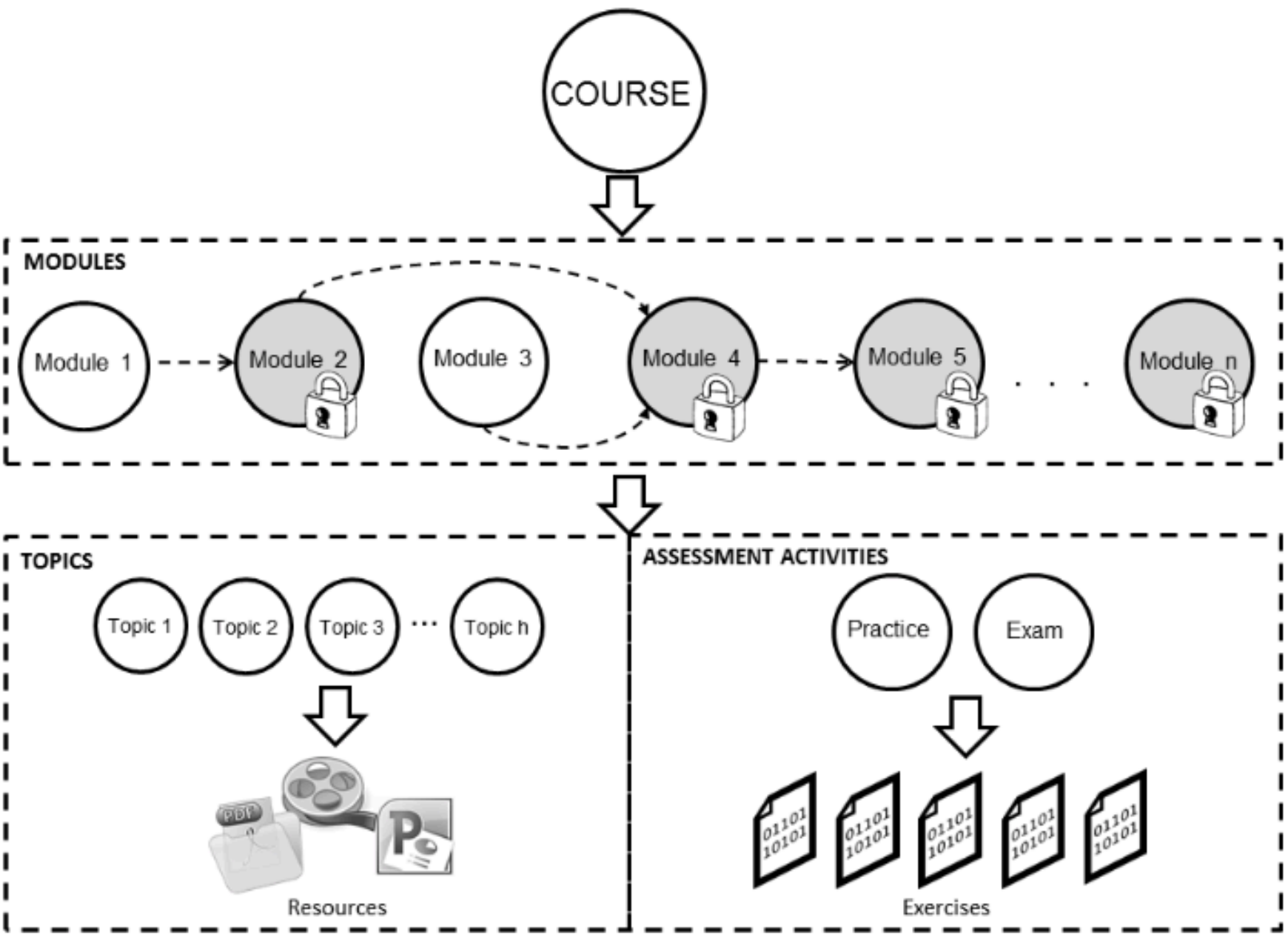
engineering course where students should work in on a programming project simulating a working day in the industry. Every week students received a challenge regarding a particular extreme programming practice. Authors concluded that using this technique the students' engagement was improved, as well as their focusing and general performance.

Then, considering all the evidence in favor of competitive programming and gamification, what we try to present in this paper is an integrated model that considers both, tightly integrated, within a Learning Management System (LMS) specially designed for it.

2. Course design and implementation

The first step to incorporate the aforementioned approaches is to determine a well-defined course structure to support them in a fashion that fits teacher curricular requirements. Such a structure, shown in Figure 1, defines the course as the root of a hierarchical arrangement of modules, which are understood as thematic units. Those modules may or may not have prerequisites among them, i.e. the teacher determines which modules must be fulfilled first to access other modules. This restriction, when the course design defines so, aims to face a problem that usually occurs in programming courses: students with conceptual gaps in previous topics often struggle when learning new ones.

Figure 1
Course structure



It does not mean that all courses must follow a linear progression. In fact, such a structure allows for students moving forward through all modules whose prior knowledge has been already passed or has none at all. This way, students may acquire expertise moving in several directions, but guarantees a proper development of the required skills.

Each module is composed of a set of related topics. In a computer science 101 course, for example, a module like "Arrays" may be composed by of topics such as "Unidimensional arrays" and "multidimensional arrays". In the same way, in a data structures course, a

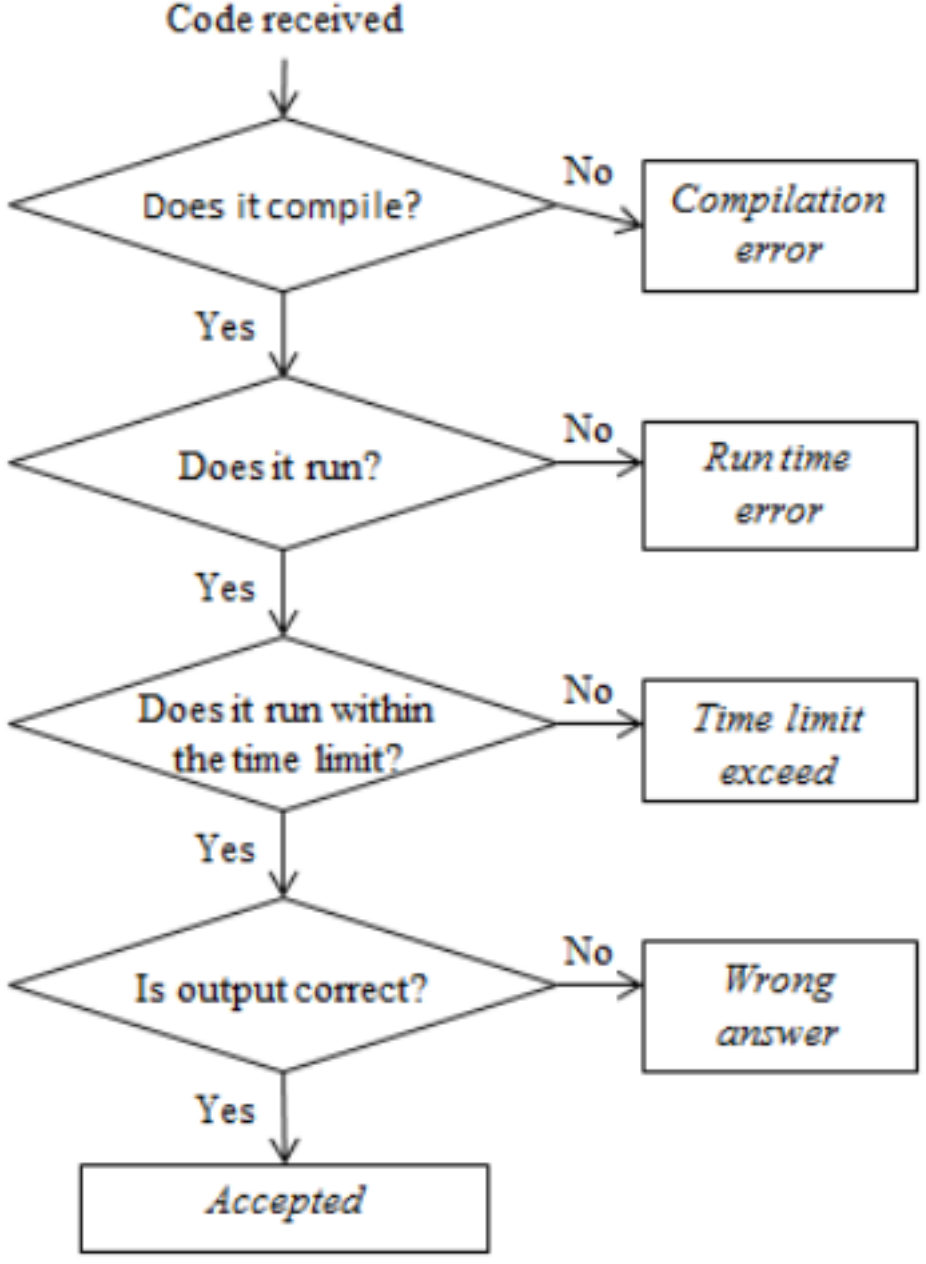
module like “Linked lists” may be composed by of topics such as “Single linked lists”, “Double linked lists”, “Circular linked lists” and so on. To cover those topics, teachers can use resources, e.g. video lectures, slides, documents, etc. Now, for assessment of the modules, the teacher can use two kinds of activities: practices and exams. The difference between the two is the lifespan of each. Practices are medium term activities, often spanning days or weeks. Exams for the other side are short term activities, usually spanning minutes or hours. In both cases, they are composed of a set of exercises.

These two activities are the ones that determine if a module has been successfully achieved or not and therefore if a determinate prerequisite has been passed or not. For doing so, the teacher must define a passing grade for each module in terms of the minimum percentage of completed exercises.

Once the course structure has been set, the competitive programming strategy can take place. The way for doing so is through the exercises that compose the assessment activities. The trick with those exercises is that they follow the ACM-ICPC standard, and the corresponding student solutions, i.e. codes, can be automatically verified by an automatic judge. Such a standard defines six elements (all mandatory and always in the same order) which exercises must contain: 1) a title, often flashy, of the exercise; 2) the time or/and memory restrictions; 3) the description, detailed enough and often ludic, of the problem being faced; 4) the explicit explanation of the input data, including quantity, format and values range; 5) the explicit explanation of the output including the format in which the code results must be displayed; and 6) an input sample with its corresponding output. Tens of thousands of exercises following this standard may be found in Universidad de Valladolid’s online judge site (https://uva.onlinejudge.org), one of the most popular platforms for competitive programming worldwide. For instance, the exercise 1001, “Say Cheese”, may be found in https://uva.onlinejudge.org/external/10/1001.pdf.

What this standard facilitates is the automatic verification of codes. This process is made through what is known as a judge, usually online, which is in charge of compiling and executing the codes sent by students. This is done using a data set that fulfills input and output rules according to the exercise definition but that are different, generally a lot larger or trickier, than those presented in the input and output samples. According to the code response, the judge may provide one of five possible feedbacks as shown in Figure 2.

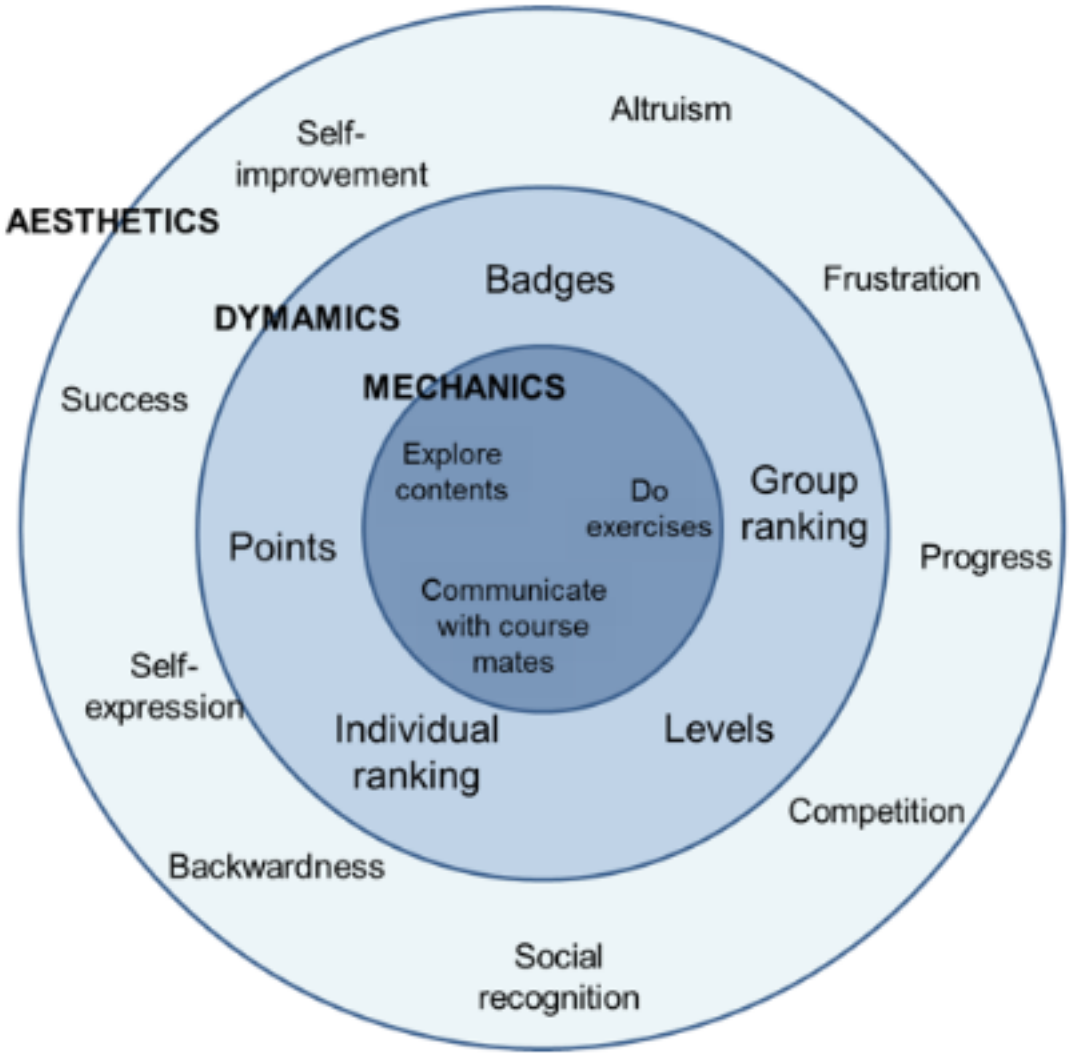
Figure 2
Codes judge feedbacks



Now, about the second element of the strategy, i.e. gamification, the goal was to integrate game elements into the course design to foster interest and extrinsic motivation in students. For doing so, the framework MDA (Mechanics, Dynamics and Aesthetics) described by Hunicke, Leblanc, and Zubek (2004) was adopted. The mechanics are the set of game rules that define the plausible actions for all participants, as well as the underlying processes that such actions trigger. Dynamics describe how those rules perform in motion, responding to the participants' actions, and working in concert with other rules. In practical terms, dynamics define the run-time behaviors of the game. Finally, the aesthetics refer to the player experience of the game. In other words, the emotional responses evoked by the Dynamics. These three components are strongly related. From the designer perspective, the mechanics generate dynamics which generate aesthetics. Although, from the player perspective, it is the other way around: players experience the game through the aesthetics, which are provided by the game dynamics, which emerged from the mechanics.

In this proposal, the corresponding instantiation of these components is summarized in Figure 3. Regarding the mechanics, there are three main actions that students may perform. First, students may explore resources, i.e. they can access all resources that the teacher, or even another student, has posted in the course. They may also provide feedback about those resources rating and commenting them. Second, students may submit exercises, but only if the corresponding timespan is underway and if the corresponding prerequisites have been accomplished. In the model proposed students may submit their codes as many times as necessary for the same exercise. Every submission is verified by the automatic judge and the corresponding response is delivered as shown in Figure 2. Third, students may communicate with their classmates. Learning is a social process and, to ease that, several means must be provided, both synchronous and asynchronous. Some tools for doing so include chat, mail, forums, and newsfeed channels.

Figure 3
Instantiation of the MDA model



As for the mechanics, five of them are proposed. First, the points provide a quantitative measure of the efficiency of the student solutions to exercises. As explained before, an exercise is successfully solved only when the automatic judge returns an "Accepted" response; in all the other cases a solution is considered as insufficient. As it was also explained previously, a student can submit as many solutions for a given exercise as necessary. It does not mean that all submissions are worth the same. In fact, the points that a student obtains in an exercise are calculated as a decreasing function of unsuccessful attempts. Second, the rankings are closely related to points and provide an explicit comparison of performances among students. In the individual case, it results from the descending ordered list of the sum of all points of each student. The more the points, the better the rank. Similarly occurs with the groups' rankings with the difference that the points of a group result of a function, a bit more complex than just the average but with the same aim, of the individual points of students who compose the corresponding group. This way, collaboration among group mates is promoted as well as healthy competition among "opponents". Fourth, the levels provide an objective and measurable scale of student progress in the course. Depending on some rates of exercises completion, students pass from "rookies" to "initiated", to "apprentice", and so on, until they gain the "expert" level. Fifth, the badges are distinctions given to students when they perform certain actions, or set of actions, usually as rewards for their efforts and as a motivation to keep doing them.

Finally, regarding aesthetics, Figure 3 presents some of the expected emotional responses that the aforementioned dynamics may trigger. Badges, the fifth mechanic, are precisely designed with the aim of promoting, along or complementarily to the other four, those that are positive. For example, when a student submits three solutions consecutively, all with Accepted responses, he/she earns the badge "Three straight". Similarly occurs with the "Five straight" and the "Eight straight" badges, all prompting for a success emotion. Another example is the "Altruist" badge, which is earned when a student helps others in forums and receives a minimum of 5 "likes". In total, more than 25 badges were designed.

Now, in order to materialize the model proposed, an online platform called CPP was developed. Its name obeys to the acronym in Spanish for Programming Practice Center, but also honors the famous programming language C++ created by Bjarne Stroustrup (Usually the extension for a C++ code is .cpp due to C plus plus). CPP may be considered as a Learning Management Systems (LMS) in the sense that it does not only provides course

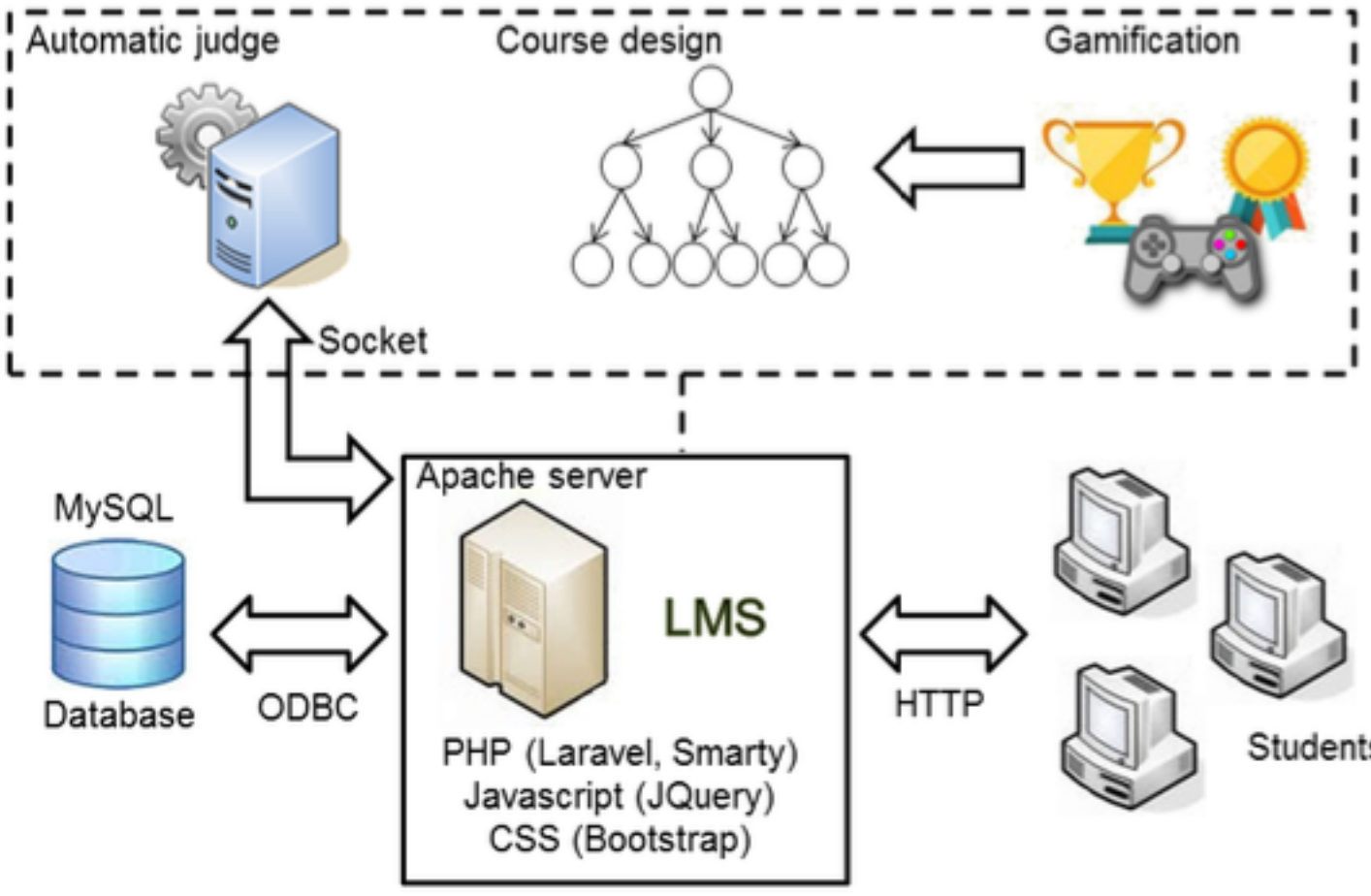
contents but also keeps track of students' progress.

As shown in Figure 4, CPP has a client-server architecture implemented using various open source tools and languages. The web server uses APACHE which compiles and executes PHP files and connects to the MySQL database. Such a server connects to the automatic judge that may be hosted on the same machine, in a separate one, or even distributed in an array of servers. Every time a student submits a code, the web server sends a request to the judge which creates a thread to handle each submission simultaneously and independently using JavaSockets.

From the client side, the use of a web browser is necessary to access the platform. It uses HTML5 for layout and Bootstrap CSS framework for styling. It also uses Javascript, JQuery and AngularJS libraries for different dynamic actions that take place in the DOM.

All CPP components were implemented under the framework Laravel using the View-Controller model. Such a framework facilitates various functions such as user authentication, active record, Object-Relational Mapping (ORM), automatic routing, template engines, RESTful web services, among others.

Figure 4
CPP architecture



Now, from the student point of view, CPP provides two main tabs: course content, and student profile. As shown in Figure 5, the course content tab allows students for navigating through the course modules depending on the prerequisite structure. Once inside a module, students have access to resources, practice exercises, and exam exercises. In the case of resources, students can view them directly or download them and, if they want, can also leave a comment or give a rate. Such a rate in CPP is represented by an integer scale between one and five stars.

Figure 5
Course modules view

In the case of the exercises, students can read their descriptions according to the ACM-ICPC standard explained earlier. Once read, they can submit solutions to be verified for by the automatic judge. For doing so, students must indicate which of the two available languages (C++ and JAVA) they used, and then they can upload their code or paste it into the screen. Before the actual submission, students also have the option of testing codes with their own test cases. This process helps them to foresee any compilation or even runtime errors that codes could have. Finally, when students conclude a submission, CPP returns them a unique identifier to associate it with the future response that will be delivered once execution in the automatic judge finishes. In the case of an Accepted response, CPP would also show the points obtained and the execution time.

The student profile tab (see Figure 6) is where students can view all the information about their progress: badges obtained, exercises solved, modules unlocked, submissions statistics, points won, and position in the ranking.

Figure 6
Student profile view

Other important LMS features were also implemented in CPP: a chat which allows for synchronous communication, a forum manager to create discussions to solve doubts, a private messaging system for asynchronous communication, and a notifications panel to inform students about interesting actions that have occurred to them while were offline.

Teachers also have an important role in CPP. Besides all the obvious interfaces for course-editing, teachers can see detailed information about the course both globally and individually. Among other things, CPP allows for students' progress monitoring, submissions timeline, acceptation percentage, logging times, etc.

---

3. Results

With the aim of validating the model proposed as well as the corresponding platform, a study case was conducted in the *Institución Universitaria Politécnico Gran Colombiano*, in the city of Medellin, Colombia. In total, 43 freshman engineering students participated, all of them attending the course "algorithmic thinking". Although the title could sound a little misleading, such a course may be considered as a "computer programming 101".

A quasi-experimental and correlational design was followed using two groups, control and experimental. The control group had 24 students, 15 male and 9 female, whereas the experimental group had 21 students, 18 male and 3 female. With few exceptions, all most students were in their early 20's. Selection of students into both groups was not made in a random way. Instead, the groups' composition depended entirely on the institution registration process, and therefore was not biased in any way by the experiment. Both groups shared the exactly same settings: course syllabus, lectures, teacher, hourly load, etc. The only difference between the two was that experimental group used CPP in addition to the face-to-face classes whereas the control group did not. Instead, control group used traditional workshops.

The aim of the study was to determine if there was any difference in the academic performance between the two groups after the course conclusion. But first, a pre-test was applied to determine the initial conditions of both. Such a test was about mathematical logic, a fundamental skill for computer programming learning (Hwang et al., 2012). The reason for

doing so instead of using a specific test about computer programming was that all students reported no previous experience with programming. Table 1 presents the descriptive statistics of such a pre-test considering a [0, 5] scale with one decimal digit. These results suggest that there is no previous initial difference between both groups (Mann-Whitney U = 281.0, p> .05).

Table 1
Pre-test results

Group	Mean	Median	SD	Skewness	Kurtosis
Control	3.63	3.6	.97	-.28	-.47
Experimental	3.45	3.5	.96	.07	-.77

Three post-tests were also performed, one for each module in the course. Table 2 presents the descriptive statistics of the three of them. There was a fourth, introductory module in the course but it did not have a test. In all three cases, Conditionals (Mann-Whitney U = 353.0, p< .05), Iteration (Mann-Whitney U = 352.0, p< .05), and Arrays (Mann-Whitney U = 342.5, p< .05), students in the experimental group performed better than those in the control group.

Table 2
Post-test results

Module	Group	Mean	Median	SD	Skewness	Kurtosis
Conditionals	Control	3.95	3.8	.72	-.63	1.46
	Experimental	4.26	4.3	.53	-1.54	2.32
Iterations	Control	3.66	3.7	.36	.31	.42
	Experimental	3.99	3.9	.54	1.43	1.89
Arrays	Control	3.62	3.7	.42	.54	.32
	Experimental	3.95	3.8	.56	0.14	-0.74

A perception survey was also administrated to the experimental group asking about the course model used. It consisted of four questions/statements that must be answered using a Likert scale of five points: 1=Strongly disagree, 2=Disagree, 3=Neither agree nor disagree, 4=Agree, 5=Strongly agree.

> Question 1 (Q1): I feel that the model used in this course is more effective, i.e. my academic performance was better, than if a traditional course model were used.
>
> Question 2 (Q2): The course structure (division into modules, resources, and exercises) seemed appropriate.
>
> Question 3 (Q3): The automatic verification and feedback provided when submitting the exercises were an aid in the learning process.
>
> Question 4 (Q4): Game dynamics, such as points, ranking, and badges, motivated me to study harder.

The first question was related to the course model as a whole, whereas the remaining three

refers to its specific components: course structure, competitive programming, and gamification strategies. The descriptive statistics of the corresponding responses are presented in Table 3. These results suggest that most of the students felt pleased about the model proposed. Low values of the standard deviations also indicate that opinions were relatively uniform.

Table 3
Perception survey responses

Question	Mean	Median	SD	Skewness	Kurtosis
Q1	4.38	4	.59	-.29	-.61
Q2	4.67	5	.58	-1.59	1.89
Q3	4.48	5	.68	-.96	-.10
Q4	4.48	4	.51	.11	-2.21

Besides the quantitative results of the tests and the survey, students in the experimental group were also asked about their personal, qualitative, opinion regarding the experience with CPP and the underlying model. Most responses indicated the enthusiasm they felt during the course. In particular, two issues in responses were recurrent: the immediate feedback they received when submitting an exercise, and the "competition atmosphere" that pushed them to give their best effort. Some of the responses are presented below.

"*The use of a didactic platform during learning was a big help in the algorithmic thinking course.*
It encouraged competition among classmates in a very harmonious way" –
Male, 20 years old.

"*This kind of experience makes the learning process a lot more dynamic and effective.*
It should be used not only in computer sciences but also in other areas" –
Male, 22 years old.

"*I think the study methodology was interesting, especially the platform that allows us, as students,*
for getting used to programming concepts through competition within our group.
Moreover, it was a funny way of learning while practicing" –
Female, 24 years old.

4. Conclusions

As we stated in the very beginning of this paper, teaching computer programming is not an easy task. Therefore, and as an aim to deal with this problem, we presented a model to design and implement courses based on competitive programming and gamification. Experimental results during a study case suggested tangible benefits of both approaches when working together.

For the one hand, competitive programming allows for addressing many of the problems presented by students related to the lack of practice but especially to the lack of an early and proper feedback. For doing so, it takes advantage of an automatic code validator, or judge, which evaluates all students' submissions. In this way students do not need someone else, usually the teacher, to validate their solutions, letting them to perform an autonomous practice. This also allows teachers for intervening only in specific situations as well as providing more custom accompaniment for those students who need it the most.

For the other hand, when combined with gamification, more benefits arise. First, being aware of self-performance through points and rankings allows students for being constantly

challenged to surpass, not only themselves, but their peers. Second, it encourages students to be more cautious, i.e., even if they have the chance of submitting as many solutions as needed, they try to lose as few points as possible. This way they also get used to thinking in software development as an iterative process of designing and testing solutions until they work completely. Some other gamification elements also promote self-improvement. For example, when the execution times are presented to students, they usually focus on analyzing differences among their solutions to learn best coding practices or even new algorithms. Badges for the other side also serve as recognition for the student efforts in other facets. This is particularly important to promote collaboration or even student feedback to course contents, which are usually hard to witness with traditional practices.

The brief survey administrated to students also confirms those benefits. After completing the course, students who used CPP felt more confident with their knowledge and most of them agreed that learned more than if a traditional method were followed. In fact, students showed interest in using the model in succeeding courses.

Acknowledgments

This research was partially supported by the *Departamento Administrativo de Ciencia, Tecnología e Innovación (Colciencias),* Colombia, under the program *Jóvenes Investigadores.*

Bibliographic references

Akpolat, B., Slany, W. (2014). Enhancing Software Engineering Student Team Engagement in a High-Intensity Extreme Programming Course using Gamification. In *Proceedings of 27th IEEE Conference on Software Engineering Education and Training* (pp. 149–153). Klagenfurt, Austria.

Bennedsen, J., Caspersen, M., Kölling, M. (2008). Reflections on the Teaching of Programming, *Lecture Notes in Computer Science*, 4821, Springer-Verlag: Berlin.

Combéfis, S., Wautelet, J. (2014). Programming Trainings and Informatics Teaching Through Online Contests. *Olympiads in Informatics*, 8, 21-34.

Coles, D., Jones, C., Wynters, E. (2011). Programming contests for assessing problem-solving ability. *The Journal of Computing Sciences in Colleges*, 28(3), 28-35.

Deterding, S., Khaled, R., Nacke, L., Dixon, D. (2011). Gamification: Toward a definition. In *Proceedings of the CHI 2011 Workshop Gamification: Using Game Design Elements in Non-Game Contexts*. (pp. 1–4). Vancouver, Canada.

Garcia-Mateos, G., Fernandez-Aleman, J. (2009). Make Learning Fun with Programming Contests. *Lecture Notes in Computer Science*, 5660, 246-257.

Halim, S., Halim, F. (2010). Competitive programming in National University of Singapore. In Verdú, E., Lorenzo, R., Revilla, M., Requeras, L. (Eds.), *A New Learning Paradigm: Competition Supported by Technology* (pp. 173–206). Madrid: Sello Editorial.

Hunicke, R., Leblanc, M., Zubek, R. (2004). MDA: A Formal Approach to Game Design and Game Research". In *Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence* (pp. 1–7). San José, California.

Hwang, W.-Y., Shadiev, R., Wang, C.-Y., Huang, Z.-H. (2012). A pilot study of cooperative programming learning behavior and its relationship with students' learning performance. *Computers & Education*, 58(4), 1267-1281.

Ibanez, M., Di-Serio, A., Delgado-Kloos, C. (2014). Gamification for Engaging Computer Science Students in Learning Activities: A Case Study. *IEEE Transactions on Learning Technologies*, 7(3), 291-301.

Koffmann E., Brinda T. (2003). Teaching Programming and Problem Solving: The Current State. In Cassel L., Reis R.A. (eds) *Informatics Curricula and Teaching Methods. IFIP — The International Federation for Information Processing, vol 117* (pp. 125-130). Boston: Springer.

Lahtinen, E., Ala-Mutka, K., Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. In *ITiCSE '05 - Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* (pp. 14–18). Caparica, Portugal.

Li, C., Dong, Z., Untch, R., Chasteen, M. (2013). Engaging Computer Science Students through Gamification in an Online Social Network Based Collaborative Learning Environment. *International Journal of Information and Education Technology*, 3(1), 72-77.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. (2001). A multinational, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin*, 33(4), 125-180.

Mcgettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, G., Mander, K. (2005). Grand challenges in computing: Education—a summary. *The Computer Journal*, 48(1), 42-48.

Milne, I., & Rowe, G. (2002). Difficulties in Learning and Teaching Programming — Views of Students and Tutors. *Education and Information Technologies*, 7(1), 55-66.

Mow, C. (2008). Issues and Difficulties in Teaching Novice Computer Programming. In Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education (pp. 199–204), Netherlands: Springer eds.

Renumol, V., Dharanipragada, J., Jayaprakash, S. (2010). Identification of Cognitive Processes of Effective and Ineffective Students During Computer Programming. *ACM Transactions on Computing Education*, 10(3), 1-21.

Revilla, M., Manzoor, S., Liu, R. (2008). Competitive Learning in Informatics: The UVa Online Judge Experience. *Olympiads in Informatics*, 2, 131-148.

Robins, A., Rountree, J., Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137-172.

Schoech, D., Boyas, J., Black, B. & Elias-Lambert, N. (2013). Gamification for Behavior Change: Lessons from Developing a Social, Multiuser, Web-Tablet Based Prevention Game for Youths. *Journal of Technology in Human Services*, 31(3), 197-217.

Wegner, P., Roberts, E., Rada, R., Tucker, A. (1996). Strategic directions in computer science education. *ACM Computing Survey*, 28(4), 836-845.

---

1. Associate professor, PhD. Computer and decision sciences department. National University of Colombia. jmoreno1@unal.edu.co

2. PhD student. Computer and decision sciences department. National University of Colombia. afpinedac@unal.edu.co

---

Revista ESPACIOS. ISSN 0798 1015
Vol. 39 (Nº 35) Year 2018

[Index]

[In case you find any errors on this site, please send e-mail to webmaster]