REDEFINING THE FRAMEWORK FOR TEACHING PROGRAMMING TO PRIMARY SCHOOL STUDENTS: RESULTS FROM THREE PILOT PROJECTS

Emmanuel Fokides and Pinelopi Atsikpasi
University of the Aegean
Greece

Abstract

The study summarizes the findings of three pilot projects in which 2nd, 5th, and 6th-grade primary school students were taught basic programming concepts using game-like applications. In all projects, two more groups of students were formed that were taught the same subjects using conventional methods. Results' analyses revealed that students who used the game-like applications had better learning outcomes compared to the ones that were taught conventionally. The results can be attributed to increased students' motivation and to the applications' game-like characteristics. Based on the results, suggestions for redefining the framework for teaching programming are presented.

Introduction

Technology has changed many aspects of our lives. As far as education is concerned, technology has imposed a significant shift in focus: from knowledge acquisition to the acquisition of a set of skills that will render students creative and capable of responding to the needs of modern society. Students must stop being passive users of devices and applications and become content designers and creators through them (OECD, 2015). Even if Prensky (2001) describes young students as *digital natives* because of their familiarity with technology, their skills are still associated with the simple use of devices and applications. The prevailing educational model continues to be that of facilitating learning through the use of technology, that is also related to the simple use of ICT tools during teaching.

The question emerges: How can we turn students from adept users to skillful content designers and creators through technology? There are many who believe that this can be achieved if students acquire programming knowledge and skills (Resnick et al., 2009). There are multiple benefits for students when they learn how to program: development of analytical thinking, development of skills related to the design of algorithms, and a positive impact on their creativity and imagination (Fessakis, Gouli, & Mavroudi, 2013; Liu, Cheng, & Huang, 2011). Researchers suggested that when the teaching of programming becomes an enjoyable experience the results are noteworthy (e.g., Margulieux, Guzdial, & Catrambone, 2012).

The teaching of programming concepts in Greek primary schools is included in the last two grades, not as an independent course, but within the IT course (Hellenic Ministry of Education, 2003). However, the content is poor, outdated, not well organized, and students face difficulties (Papadakis,

Orfanakis, Kalogiannakis, & Zaranis, 2014). Therefore, a two-fold intervention is needed to rectify the problem. The first is to redefine the objectives and the content of programming as a teaching subject. The second is to find easy and fun to use tools, so students are motivated to learn how to program and to develop positive attitudes towards this subject. Three pilot projects were designed and implemented over a school year. Though the target groups were students of different ages, they shared some common features: (a) the tools that were used exploited the elements of fun and play, (b) the programming concepts that were taught were basic but went beyond those included in the official curriculum, and (c) the duration and sample sizes were sufficient so that reliable conclusions can be drawn. The main research questions were: (a) to what extent primary school students can understand basic programming concepts, (b) what is the appropriate teaching method, (c) how important are the elements of fun and play, and (d) what is the role of students' autonomy during the learning process. The coming sections summarize the rationale, methodology, and findings of these projects. On the basis of the experience gained, specific suggestions on how to improve the current situation are also presented.

Programming as a Teaching Subject

Programming, as a teaching subject, is included in the Greek primary school's curriculum, as part of the Informatics' course, which is taught just for an hour each week, and only in the last two grades. Its objectives are that through the use of a simple programming language (Logo-like) students learn how to use simple commands in order to create shapes or solve simple problems, understand algorithmic structures, and develop their problem-solving skills (Hellenic Ministry of Education, 2003). Apart from the fact that Logo is outdated, compared to other modern programming languages for children, the curriculum is poor both in terms of its duration and content (Grigoriadou, Gogoulou, & Gouli, 2002). In general, students face some major problems when they learn how to program. They have a poor understanding of how programs are executed (Pea, 1986), and of the rules, logic, and syntax of the programming languages (Kristi, 2003). Variables, as well as other concepts, are not easy to grasp (Pane & Myers, 1996). The reasons given for the above issues are young students' lack of logical reasoning and their undeveloped algorithmic and critical thinking (Robins, Rountree, & Rountree, 2003).

The teaching/learning of programming fosters a series of mental and cognitive skills. Besides learning fundamental programming concepts (Zhang, Liu, Ordóñez de Pablos, & She, 2014), students can develop a positive attitude toward learning computing in general (Fessakis et al., 2013). A better understanding of mathematical concepts and improvement of their social skills (Fessakis et al., 2013), problem-solving skills (Akcaoglu & Koehler, 2014), computational thinking (Grover & Pea, 2013), higher order thinking skills (Kafai, Burke, & Resnick, 2014), as well as an impact on their creativity and imagination (Liu et al., 2011), were noted. There is extensive literature on the ways that programming can be taught to primary school students. For example, Scratch and its versions attracted the attention of the scientific community (e.g., Su, Huang, Yang, Ding, & Hsieh, 2015). Many have pointed out that its effectiveness is the result of its game-like characteristics (e.g., Su

et al., 2015). Furthermore, very positive results yield programming environments where their purpose is the development of games. Besides having the positive effects that were previously mentioned, research has shown that such programming environments render students more creative and more motivated for learning how to program computers (Preston & Morrison, 2009).

The Pilot Projects

Assuming that game-like programming environments are particularly effective, the interest turned to them, and it was decided to examine their effectiveness. Therefore, in 2016, three pilot projects were designed and implemented. In the sections that follow, brief summaries of their rationale and methodology, as well as their main conclusions, are presented.

Pilot Project 1

In this project, the target group was sixth-grade students (ages 11-12). The tool chosen for teaching programming was Microsoft's Kodu (http://www.kodugamelab.com/), which allows the rapid development of 3D games. The programming language has very simple rules and it is based on physical terms and concepts such as see, hear, and bump, for the control of the games' characters and objects. Even 10-year-olds developed their own games. The programming concepts that were taught were variables, sequences, and subroutines. Two two-hour sessions were allocated for the teaching of each programming concept. Students worked in pairs. The teacher introduced each programming concept and students were then asked to develop mini-games using the programming concept that had been introduced to them. To enable comparison of the results, two more groups of students were formed. To the first, only evaluation sheets, presented in the coming paragraph, were administered. Thus, it was examined what students can intuitively perceive regarding the above programming concepts. The second group was taught conventionally. The teacher taught using notes, presentations, brochures, and the whiteboard. Instead of the students developing mini-games, the teacher posed problems, derived from students' everyday life, associated with each programming concept, and students (working in pairs) solved them, on paper, in the form of pseudocode. For example, in sequences, they were asked to write down the recipe for a pizza in a form of a sequence of events.

The assessment of students' performance was done using: (a) evaluation sheets that were given immediately after the end of each session and (b) delayed post-tests, that were given about two weeks after the end of the project, to test the sustainability of knowledge. Each of the above tests consisted of two distinct sections. The first had multiple choice, fill-in-the-blanks, and right-wrong questions (at least 10 of them). In the second part, students were instructed to transcribe, using programming terms and concepts, everyday life activities (at least 5 such problems). Also, at the end of the project, a short questionnaire was administered in order to investigate the attitudes and opinions of students for Kodu (15 Likert-type questions). A total of 66 students participated in this project coming from three neighboring schools in Athens, Greece. The analysis of the results (available at http://opensimserver.aegean.gr/pilotproject1.htm) revealed that the group of

students that was taught using Kodu achieved statistically significantly better learning outcomes compared to the other groups, in all cases. The vast majority of students reported that they liked working with Kodu a lot. At the same time, they stated that the whole process was pleasant, fun, and like a game. No problems whatsoever were reported.

Pilot Project 2

In the second project, the target group was fifth-grade students (ages 10-11). The research methodology was different than the previous one. Since conventional teaching did not produce good results, it was decided to examine different types of teaching methods, but, in all, collaboration between students played a major role. Again, three groups of students were formed. All used Kodu, and students worked in groups. In the first, the teacher had an active role, systematically teaching the programming concepts, by giving examples in Kodu, and by providing constant support to students. In the second, the teacher had only a supporting role (e.g., answering technical questions), and students studied the programming concepts using detailed notes. In the third group, the role of the teacher was again limited, and the notes were not available to students; they had to seek by themselves solutions to the problems they faced. The main goal was for students to develop a complex game by the end of the project. This was implemented in three stages. First, students were asked to develop simple games, without any programming, in order to explore the objects included in Kodu. The second stage involved the development of a simple game, by adding interactions and by implementing a simple game scenario that was given to them. Students encountered important programming concepts such as variables, sequences, logical expressions (AND, OR), conditions (When-Do), and subroutines. In the final stage, a detailed game scenario was given to students, and they were asked to implement it in the best way they could. This stage was significantly longer, compared to the previous stages. The project lasted for about three months (70 hours for each group, 6 hours per week), due to the complexity of the tasks together with the need to provide students enough time to understand all the programming concepts and to be able to apply them. The target group was 63 fifth-grade students coming from the same schools as in the previous project.

Research data was collected by evaluating students' games. For their evaluation, the technique of content analysis was utilized (conducted by three independent raters), and a complex scoring system was developed, containing both quantitative and qualitative criteria. The quantitative criteria included the number and types of commands used, if they were used properly, and if there were any programming errors. The qualitative criteria were those proposed by Consalvo and Dutton (2006), for example, the aesthetic integrity of the game, the complexity of the levels, the complexity of commands, the gameplay, etc. In addition, at the end of the project, a short questionnaire was administered in order to examine the attitudes and opinions of students regarding Kodu (15 Likert-type questions). The data analysis (available at http://opensimserver.aegean.gr/pilotproject2.htm) revealed an interesting finding. The teaching method did not have any statistically significant impact on students' scores. Also, all groups liked this programming environment,

their enjoyment was high, the development of games seemed easy and like a game.

Pilot Project 3

The last study examined if it is possible to teach programming to younger ages than the official curriculum dictates. The target group was second-grade students (ages 7-8). Because Kodu could not be used by children of this age, tablets and an application, namely Kodable (https://www.kodable.com/), were used. Kodable was selected because of the simplicity of its interface, the game-like features, and the existence of ready-made and detailed lesson plans. Although it is in English, the interface can be easily understood rendering knowledge of English unnecessary. The student/user guides the application's character through labyrinthine levels, collecting as many coins as possible. Each level is completed when the character reaches the exit. The guidance is done by using the available commands as many times as the user wants. The commands are placed by dragging and dropping them to a limited number of empty slots, suggesting that the program must be completed using a limited number of commands. The user executes the program, sees the results, and, in case of an error, he/she can redo the programming. The levels are of escalating difficulty (e.g., more complex paths, fewer available commands). It is worth noting that there is no single correct solution to each level. Sequences, conditions (if/then) and loops were taught using this application. The lessons' plans and activities were translated and adapted into Greek. At the beginning of each session, the teacher made a short introduction about the programming concept that he was about to teach, drawing examples from students' everyday lives. Next, students worked, in pairs, using the tablets, resolving the levels of the corresponding concept. In-classroom activities followed, which required teamwork and included worksheets and games. Each session lasted for two teaching hours, and each programming concept required two sessions.

Immediately following the end of the teaching of a programming concept, students completed an evaluation sheet, consisting of three distinct parts. The first one had multiple choice, fill-in-the-blanks, and right-wrong questions. In the second part, students were instructed to transcribe, using programming terms and concepts, everyday life activities (as in the first pilot project). The third part followed Kodable's philosophy and presentation layout. Students were presented with a level, and they had either to complete the missing commands or to check whether the solution was correct (identifying any errors). Also, about a month after the end of the project, students completed a delayed post-test which had the same structure as the evaluation sheets but included all the programming concepts that they were taught. They also completed a short questionnaire for the evaluation of their experiences and views regarding the use of tablets/application (15 Likert-type questions).

For examining the significance of the project's results, two more groups of students were formed. The first one used board games instead of tablets. This method has been used by other researchers with noteworthy results (e.g., Mavridis, Siribianou, & Alexogiannopoulou, 2015). Each board game was a printed and enlarged Kodable's level. The same was done for the characters and for all the other elements included in the application. The students,

working in pairs, placed the various elements/commands on the board and the teacher "executed" the "program" determining if it "worked" properly. The in-classroom activities, as well as the way students worked, were the same as in the tablets group. The second group of students was taught conventionally, using notes. These notes followed Kodable's philosophy and way of presenting the learning material. Once again, students worked in pairs. The in-classroom activities were also the same as in the previous groups. The sample size was 69 second-grade students coming from the same schools as in the previous projects. The data analysis (available at http://opensimserver.aegean.gr/pilotproject3.htm) indicated that the teaching of programming concepts using Kodable produced statistically significantly better learning outcomes in all cases except sequences, where the group that used Kodable and the team that used the board game had the same results. According to students' responses, conditions was the most interesting programming concept, followed by sequences and loops, while conditions were considered the most difficult one. In addition, they stated that they learned quite a lot and quite easily. They also found tablets easy to use and motivational. Finally, students made very positive remarks regarding their experiences while using the tablets and the application, noting its game-like features.

Discussion – Towards a New Framework for Teaching Programming to Primary School Students

Regarding Kodu, an important finding was that the results of the first study were in line with the findings of similar studies (e.g., Earp, Dagnino, & Ott, 2014; Shokouhi, Asefi, Sheikhi, & Tee, 2013). Their findings indicated that Kodu made the teaching of programming concepts more enjoyable, and, because of its game-like features, it helped students to have a better understanding of basic programming concepts and solve complex programming problems. After all, Kodu's main purpose is to develop games and games are compatible with children's mentality (Prensky, 2001). It should also be noted that students, although young, did not face significant problems while using it. Based on these findings, it can be argued that Kodu is an attractive and effective tool for teaching programming concepts to students. The third pilot demonstrated that the teaching of programming concepts, to very young students, using tablets and game-like applications, is more effective than conventional teaching methods. The results are in agreement with the existing literature that emphasizes the relationship between the use of mobile devices and the good learning outcomes regarding programming concepts (e.g., Armoni, Meerbaum-Salant, & Ben-Ari, 2015). The absence of usage problems was noted in other studies, which attributed this finding to the familiarization of -even very young- children with electronic devices (Goodwin, 2012).

As for the appropriate teaching method, one should take into consideration the results of the second pilot. It seems that the teaching method is not so important if students have enough time to study and practice. This is supported by the fact that all groups had the same learning outcomes. This finding may seem surprising and perhaps difficult to interpret. Additionally, from the literature review, no similar methodological approaches were

identified that would have allowed the comparison of the findings. But a closer examination of the results can lead to an interesting conclusion. Unlike other studies, the games, developed by the students, were examined.

Therefore, what was evaluated was not an "instantaneous" effect, like in a test or in an evaluation sheet, but the result of many hours of work, trials and errors, testing, and exploring alternative solutions. It is quite possible that, initially, the three groups had differences, but these were eliminated as students had enough time to improve their games. So, even if a teaching method was not that effective, students (and their work) were the factor that balanced the results. Consequently, one must reflect on how the students worked. The dominant element, in all three pilots, was students' collaborative work. Applying constructivist views for the learning process (Papert, 1993), students expressed and discussed their views and collaborated with each other (Ertmer & Newby, 2013). Further, they had the opportunity to actively participate in the learning process, study the subject in-depth, and discover its basic principles, as suggested by inquiry-based instruction (Dostál, 2015). Important elements in this view are intuitive thinking, logical leaps, originality, and the conception of radical solutions to problematic situations.

The results noted can be attributed to the use of tools that raised students' interest. Indeed, this is evident in their answers to the relevant questions. This finding is common to many studies (e.g., Earp et al., 2014; Goodwin, 2012; Shokouhi et al., 2013). This seems to have led to increased incentives for learning and to a better understanding of the programming concepts, which, in turn, led to better learning outcomes, as noted by other researchers (e.g., Snell & Snell-Siddle, 2013). Students had the ability to control the outcomes of their work and could easily monitor their progress, either by running their games in Kodu or by executing their programs in Kodable. Thus, they had greater control over the learning process and greater autonomy, as West (2013) pointed out. On the basis of the above, education administrators and policy makers can consider:

- The incorporation of game-like programming environments, such as Kodu and Kodable, into the curriculum in order to improve the way that programming is taught to primary school students.
- A teaching framework can be derived from an analysis of the methodology applied to the pilots: (a) students' collaboration and (b) with increased autonomy so as students to have the opportunity to discover, by themselves, their own solutions to specific programming problems.
- On the basis of the results, it can be argued that programming can be taught at a very early age.
- Programming courses should have enough time allocated to them (in terms of teaching hours), so the necessary skills can be developed.
- Finally, a greater involvement of teachers in the whole process should also be considered. Training will probably be necessary, but this is not expected to be that difficult as the proposed programming environments are simple to use and easy to learn.

Conclusion

This study summarized the findings of three research pilots that resulted from the need to examine the effects of using game-like programming environments in order to teach basic programming concepts to primary school students. Despite the positive results, there are limitations that need to be acknowledged. Although the samples were sufficient for statistical analysis, they were relatively small; thus, the results cannot be easily generalized. The inclusion of more programming concepts would have allowed a deeper understanding of the problem. Finally, students may not have been completely honest in their responses, confusing the questionnaires with some form of evaluation. Future studies could utilize larger sample sizes and include additional programming concepts. In order to have a wider range of results, both quantitative and qualitative methods (such as interviews with students and teachers and observations) can be used. The use of other programming tools would allow their comparison and could lead to the selection of other appropriate environments. Finally, it would be interesting to examine the learning outcomes when teaching programming to even younger ages. In conclusion, it can hardly be said that the subject is closed. More extensive projects, in terms of duration and content, but also with the use of other tools and teaching methods, are planned for the near future. However, the evidence, so far, supports the view that game-like programming environments have a positive impact on the learning of programming concepts, especially at younger ages.

References

Akcaoglu, M., & Koehler, M. J. (2014). Cognitive outcomes from the Game-Design and Learning (GDL) after-school program. *Computers & Education, 75*, 72-81.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "real" programming. *ACM Transactions on Computing Education (TOCE), 14*(4), 25.

Consalvo, M., & Dutton, N. (2006). Game analysis: Developing a methodological toolkit for the qualitative study of games. *Game Studies, 6*(1), 1-17.

Dostál, J. (2015). *Inquiry-based instruction: Concept, essence, importance and contribution*. Olomouc, Czech Republic: Univerzita Palackého v Olomouci.

Earp, J., Dagnino, F. M., & Ott, M. (2014). Learning through game making: an HCI perspective. In C. Stephanidis & M. Antona (Eds.), *Universal access in human-computer interaction. Universal access to information and knowledge* (pp. 513-524). Cham, Switzerland: Springer International Publishing.

Ertmer, P. A., & Newby, T. J. (2013). Behaviorism, cognitivism, constructivism: Comparing critical features from an instructional design perspective. *Performance Improvement Quarterly, 26*(2), 43-71.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5-6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87-97.

Goodwin, K. (2012). *Use of tablet technology in the classroom*. NSW, Australia: Department of Education and Communities.

Grigoriadou, M., Gogoulou, A. & Gouli, E. (2002). Εναλλακτικές διδακτικές προσεγγίσεις σε εισαγωγικά μαθήματα προγραμματισμού: Προτάσεις διδασκαλίας [Alternative instructional approaches in introductory programming courses: Teaching suggestions.] In A. Dimitrakopoulou (Ed.), *Proceedings of the 3rd Conference on ICT in Education* (pp. 239-248).

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38-39.

Hellenic Ministry of Education (2003). *Διαθεματικό ενιαίο πλαίσιο προγραμμάτων σπουδών* [Unified curricular framework]. Retrieved from http://www.pi-schools.gr/programs/depps/

Kafai, Y. B., Burke, Q., & Resnick, M. (2014). *Connected code: Why children need to learn programming*. Cambridge, MA: MIT Press.

Kristi, A. M. (2003). Problems in learning and teaching programming-a literature study for developing visualizations in the Codewitz-Minerva Project. *Codewitz needs analysis* (pp. 1-12). Tampere, Finland: Institute of Software Systems, Tampere University of Technology,

Liu, C. C., Cheng, Y. B., & Huang, C. W. (2011). The effect of simulation games on the learning of computational problem solving. *Computers & Education, 57*, 1907-1918.

Margulieux, L. E., Guzdial, M., & Catrambone, R. (2012, September). Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. *Proceedings of the Ninth Annual International Conference on International Computing Education Research* (pp. 71-78). ACM.

Mavridis, A., Siribianou, E., & Alexogiannopoulou, B. (2015) Διδασκαλία προγραμματισμού στο νηπιαγωγείο και το δημοτικό, χωρίς τη χρήση υπολογιστή [Teaching programming to kindergarten and primary school students without using a computer]. *Proceedings of the 9th Panhellenic Conference of ICT Educators*. Kastoria, Greece: PEKAP.

Organisation for Economic Co-operation and Development. (OECD). (2015). *Students, computers and learning: Making the connection*. Paris, France: PISA, OECD Publishing.

Papadakis, S., Orfanakis, B., Kalogiannakis, M., & Zaranis, N. (2014). Περιβάλλοντα προγραμματισμού για αρχάριους. Scratch & Αρρ Inventor: Μια πρώτη σύγκριση [Programing environments for novices. Scratch & Αρρ Inventor: A comparison]. *Proceedings of the 7th Panhellenic Conference in Teaching Informatics*. University of Crete, Greece.

Papert, S. (1993). *Mindstorms: Children, computers and powerful ideas* (2nd ed.). New York, NY: Basic Books Inc.

Pane, J., & Myers, B. (1996). *Usability issues in the design of novice programming system*s. Technical Report (CMU-CS-96-132). Pittsburg, PA: School of Computer Science, Carnegie Mellon University.

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research, 2*(1), 25-36.

Prensky, M. (2001). Digital natives, digital immigrants, *On the Horizon, 9*(5), 1-6.

Preston, J., & Morrison, B. (2009). Entertaining education-using games-based and service-oriented learning to improve STEM education. *Transactions on edutainment III*. Berlin-Heidelberg: Springer

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60-67.

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

Shokouhi, S., Asefi, F., Sheikhi, B., & Tee, E. R. (2013, November). Children programming analysis; Kodu and story-telling. *Proceedings of the 3rd International Conference on Advance Information System, E-education & Development*. Singapore.

Snell, S., & Snell-Siddle, C. (2013). Mobile learning: The effects of gender and age on perceptions of the use of mobile tools. *Proceedings of The Second International Conference on Informatics Engineering & Information Science* (pp. 274-281). Kuala Lumpur, Malaysia: The Society of Digital Information and Wireless Communication.

Su, A. Y., Huang, C. S., Yang, S. J., Ding, T. J., & Hsieh, Y. Z. (2015). Effects of annotations and homework on learning achievement: An empirical study of Scratch programming pedagogy. *Journal of Educational Technology & Society,* 18(4), 331-343.

West, D. M. (2013). *Mobile learning: Transforming education, engaging students, and improving outcomes*. Washington, DC: Center for Technology Innovation, The Brookings Institute.

Zhang, J. X., Liu, L., Ordóñez de Pablos, P., & She, J. (2014). The auxiliary role of information technology in teaching: Enhancing programming course using Alice. *International Journal of Engineering Education, 30*(3), 560-565.

Authors details

Emmanuel Fokides
fokides@aegean.gr

Pinelopi Atsikpasi
premnt16002@aegean.gr