Learning programming through stepwise self-explanations

Viviane C. O. Aureliano
Federal Institute of Pernambuco
Campus Belo Jardim
Belo Jardim, Pernambuco, Brazil
vcoa@cin.ufpe.br

Patricia C. de A. R. Tedesco
Center for Informatics
Federal University of Pernambuco
Recife, Pernambuco, Brazil
pcart@cin.ufpe.br

Michael E. Caspersen
Centre for Science Education
Aarhus University
Aarhus, Denmark
mec@cse.au.dk

Abstract — **In this article we present an approach where students self-explain small pieces of code while they are studying the process of building programs from videos used as instructional material. To evaluate our approach, we carried out an experiment in which we compared it with another approach where students self-studied the same videos. Although we could not confirm that the difference between the two groups was significant, we found a positive correlation within the instructional group between participants' answers to the self-explanation questions and participants' final results. Besides that, participants provided positive feedback regarding our approach. These findings suggest our approach should be investigated in further detail, especially with regard to which instructional conditions are more effective for it.**

Keywords - Learning programming; novices; videos; self-explanations.

I. INTRODUCTION

The literature about programming education shows that novice learners experience several difficulties in acquiring programming skills. Novice programmers have fragile knowledge and, because of that, struggle when they have to apply the acquired knowledge in new situations of use [1]. Nevertheless, the major difficulty experienced by novices is how to combine and use basic structures appropriately to build a program [1][6][7].

As a way to overcome these difficulties, teachers should guide novice learners while they are teaching the programming process. One means of guiding novices is through the use of Stepwise Improvement (SI) [6][7], a framework that describes programming as an iterative and incremental process. Using instructional material structured according to this framework, novices can learn programming by developing small pieces of code in a systematic and incremental way.

However, since students do not know how to study for the needs of a programming course [25], the structure of the material alone is not sufficient to make students learn. Thus, besides the guidance regarding the structure of the instructional material, novice learners should be guided while they are learning the programming process. One evidence-based practice for improving student learning by guiding them while they are studying from some instructional material is through using self-explanations (SEs) [2][3][4]. In the programming education area, previous studies have shown SE practice is beneficial for those learners who use it [4][13][16][17][18][19].

In this context, our approach combines the SI framework with SEs in order to promote and guide novice learners' SEs while they are learning the programming process from examples presented in videos used as instructional material. Through this approach, we expect novices to study programming appropriately from these examples, and, in consequence become able to apply and use concepts that they have learned in order to implement programs that match defined requirements.

II. STEPWISE IMPROVEMENT

SI is a conceptual framework that describes programming as a systematic and incremental process that encompasses three types of activities: extension, refinement and restructuring, organized in a three-dimensional space that is explored by programmers while they are building programs [6][7]. Extension occurs when the specification is expanded in order to cover more (use) cases. Refinement occurs when abstract code is modified and becomes an executable code that implements the current specification. Restructuring occurs when an improvement of nonfunctional aspects of a solution is made (i.e., this modification does not involve a change in the apparent behavior of the solution). Figure 1 illustrates a development sequence according to SI. It consists of five ordered steps: extension, refinement, extension, refinement and restructuring.

SI provides a general framework for the characterization of the programming process. However, the primary motivation for developing the SI framework was to use it for educating novices in the skills of programming. Caspersen and Kölling [7] advocate its application in a similar way to using guided tours rather than leaving students to walk randomly on their own. In order to conduct learners' steps in the programming space offered by the framework, the authors state that teachers should be concerned with the right amount of guidance given during instruction. Using SI guarantees that important aspects of programming education are balanced: training learners' programming skills and, at the same time, keeping the cognitive load under control while students are learning.

In that sense, SI brings a significant contribution to the field of programming education. It provides guidance in the activities that it encompasses. Regarding the extension and restructuring activities, it provides guidance in the way that the instructional material is structured. Therefore, programming textbooks, assignments, lectures and examples, among other

instructional materials, can be structured using the activities defined by the framework. Regarding the refinement activity, Caspersen [7] defined an object-oriented programming process for teaching novice learners.

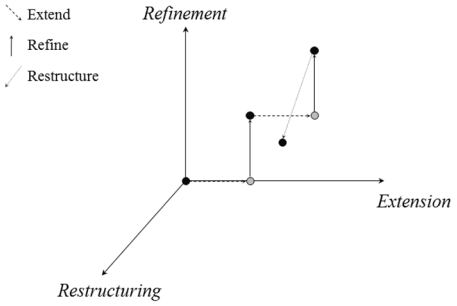

Figure 1. An example of a development sequence in SI.

III. SELF-EXPLANATIONS

The SI framework provides guidance regarding the structure of instructional material. However, it does not provide any guidance regarding the way that students can study and understand instructional material. In our approach, we propose such guidance through the use of SEs. Clark, Nguyen and Sweller [11] defined SE as *"a mental dialog that learners have when studying a worked example that helps them understand the example and build a schema from it"*. According to Chiu and Chi [10], the activity of self-explaining promotes learning through elaboration of information being studied, associating this new information with learners' prior knowledge, making inferences, and connecting two or more pieces of the given information.

The benefits of SE (i.e., SE effect) were first shown by Chi, Bassok, Lewis, Reimann and Glaser [2]. They found that good students, who self-generated a greater number of explanations while studying examples in the physics domain, scored better at problem solving when compared to poor students. Good students' explanations provided justifications for steps in the examples and related those steps to the concepts presented in the instructional material. Those students also monitored their understanding while studying the examples.

After that, the SE effect was demonstrated by many other studies in different domains, instructional contexts and types of instructional materials [3][10][11][12][14][15]. The studies by Pirolli and Recker [13] and Bielaczyc, Pirolli and Brown [4] were the first ones to demonstrate the SE effect in the programming domain. Pirolli and Recker [13] suggested that learners who could explain programs in an abstract manner or could explain the operation of the program learned better. Bielaczyc, Pirolli and Brown [4] demonstrated that students who were trained to self-explain while studying from instructional texts including examples obtained a better programming performance when compared to those students who were not trained.

More recently, some other studies have investigated the effects of SE in the programming domain. Kwon, Kulamasari and Howland [16] demonstrated the benefits of using open SE prompts while debugging web-program code in an online learning environment. Yen-Chu [17] included additional classes in a course where an instructional group learned how to self-explain computer architecture diagrams to learn assembly. These classes fostered learning of assembly and computer architecture in the instructional group. Lee, Ko and Kwan [18] added multiple choice and SE assessment levels to a programming educational game. Their findings suggested greater student engagement and understanding. Vihavainen, Miller and Settle [19] included SE questions in programming assignments and demonstrated that students who received the SE instructional material performed better than those who did not.

IV. STEPWISE SELF-EXPLANATION

Our approach is built on two lines of research: (1) the SI framework and (2) SEs. The SI framework was used to structure a set of examples presented in videos. We have chosen videos as instructional material in our approach because of their advantages. First, in contrast to textbooks, which are static, videos are an ideal instrument for showing the dynamic process of programming to novices [5]. For instance, videos play a fundamental role in showing how to use an IDE or how to implement programs incrementally, as proposed by the SI framework [5]. Second, students can manipulate those videos (i.e., play, forward or rewind) at their own pace and as many times as they need while studying programming [5]. Third, learners can watch videos wherever they are, either inside or outside the classroom. Fourth, watching videos to learn about a subject can be a very familiar activity for today's learners, also known as digital natives [22].

Since our approach is intended to teach programming to novice learners, we have used videos in which the teacher (expert programmer) performed only a sequence of extension and refinement activities. While performing an extension activity, the teacher described briefly the goal he wanted to achieve. Straight after that, while performing a refinement activity, he described how he implemented the piece of code that achieved that goal. Therefore, while learning from these videos students should achieve the intended learning outcomes (ILOs) defined from the SI framework and presented in Table 1.

TABLE I. ILOS DEFINED FROM THE SI FRAMEWORK.

Activity	Intended learning outcomes
Extension	**Describe** a goal that should be achieved in the use case being extended.
Refinement	**Apply** concepts of programming language to build an implementation that matches the goal previously described.

However, because many students do not know how to study properly for the needs of a programming course [21], the structure of these videos alone is not sufficient to make them learn. For learning to happen, it is necessary that learners understand the process of building programs presented by these videos. Thus, to help learners in understanding this material, after each extension and refinement performed by the teacher, students are invited to engage in a SE learning activity aligned with the extension and refinement activities they have just seen in the video [9]. This combination of watching the implementation of small pieces of code and then self-explaining the steps taken to implement those pieces while the

example is being built, is illustrated in Figure 2. This approach was named stepwise self-explanations (SSE), because students are explaining to themselves small pieces of code while they are being implemented incrementally in the video.

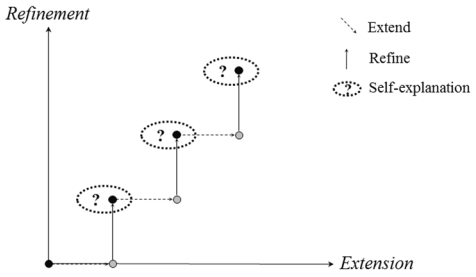

Figure 2. Stepwise self-explanation approach.

To illustrate this approach, we have chosen to use a small set of The Joy of Code videos [8] as instructional material. These videos teach Java programming language using the Greenfoot tool, and were produced according to the SI framework ideas. We present a screenshot of a video section of The Joy of Code in Figure 1. In this video section, the teacher performed an extension followed by a refinement activity. Because of that, he described the goal he wanted to achieve with the new use case he was covering with extension. Also, he described how he built an implementation that matched that goal. In this case, the goal was to make the turtle move, and the sequence of steps performed to build an implementation that matched that goal was to write the line move(1); inside the void act() method. After watching this video section, students were able to answer corresponding SE prompts related to the activities performed. These SE prompts are presented in Table 2. Similar to [4], we have defined these SE prompts using the five Ws and one H (who/what/where/when/why/how) principles of questioning.

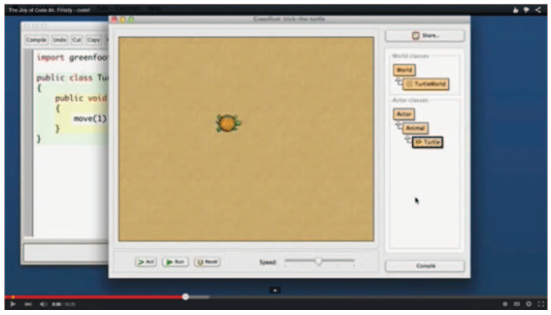

Figure 3. Screenshot of the Joy of code video recording.

The research by Pirolli and Recker [13] and Bielaczyc, Pirolli and Brown [4] also used examples as part of their instructional material, but their examples were presented to students statically and entirely on paper. Instead, the examples used in our approach are presented in videos and, because of that, they are essentially dynamic. Besides that, our approach has adopted the SI framework to structure the instructional material and the SE questions. To our knowledge, no other previous work in the programming domain has studied SE from examples presented in videos.

TABLE II. QUESTIONS RELATED TO THE JOY OF CODE VIDEO.

Activity	Questions
Extension	What is the main goal of the piece of code that he has just written?
Refinement	What were the steps/actions that he performed to achieve the goal he intended to? How did he write code to make the turtle move? What is the purpose of the parameter (number '1') in the move method? Where did he write code to make the turtle move? When did he use the move method? After he had compiled and run the code, how was the turtle behaving?

V. STUDY DESCRIPTION

The main goal of the present study was to verify whether there was any benefit in using our approach of stepwise self-explanations of examples presented in videos. To reach this goal, we have compared our approach against a traditional one, in which students studied programming from videos without any SE prompts to guide their study. Besides that, we have also verified whether a relationship exists between the learning and programming phases within the group that studied according to our approach. Thus, from the present study, we answered the following research questions:

[RQ1] Was the score of the final programming practice exercise of the instructional group greater than that of the control group?

[RQ2] Was there any correlation between the SE questions answered correctly and the score of the final programming practice exercise of the instructional group?

A. Participants

This study was carried out in a technical high school located in Aarhus, Denmark in September 2014. Fourteen second-year students of this school volunteered to participate in the study. They were divided into two groups randomly. The instructional group (SE group) studied videos using SE prompts while the control group (SS group) studied videos using their own way of studying (i.e., self-study).

B. Instructional material

As mentioned before, we used Java as a programming language and Greenfoot as a development tool. Therefore, the instructional materials used during this study were:

(i) episodes 1, 3 and 4 of The Joy of Code videos [9]. These videos were edited to make clear the boundaries between different sequences of extension and refinement activities;

(ii) the Hedgehogs, Turtle and Crab scenarios available on The Joy of Code website;

(iii) initial and final programming exercises based on the exercises from Introduction to Programming with Greenfoot [20]; and

(iv) a set of SE questions.

C. Procedure

The study consisted of five phases: (i) pre-information phase; (ii) information phase; (iii) learning phase; (iv) programming phase; and (v) post-programming phase,

performed in two sessions that took place on two different days. The first session was performed online by using a webpage and was planned to last approximately 1.5 hour. This first session consisted of the first two phases of the study, pre-information and information phases, as shown in Figure 4. The pre-information phase consisted of three activities. First, we explained the experiment to the student and collected the student's consent (or their parents' consent if they were under 18 years of age). The consent form was given previously to participants and they had to bring it back to the classroom session. Second, we collected demographic information about participants and data about their previous experience when studying from videos through an online questionnaire. A computer lab was not available for running our study; consequently we had to ask students to use their own computers. For this reason, at the end of this pre-information phase, we asked students to install in their computers Greenfoot and a software to record their computers' screens.

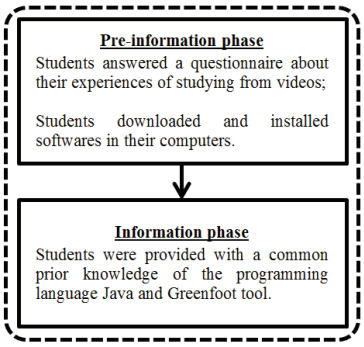

Figure 4. Online session.

The information phase consisted of two activities. First, participants had to watch sections of the episodes 1 and 3 of The Joy of Code videos. The section of episode 1 presented some examples of what students could build using Greenfoot development tool and it was used to motivate students for participating in the study. The section of the episode 3 presented some basic concepts of object oriented programming in Java, such as classes, objects and methods, and it was used to give students some prior knowledge in the subject. Students were instructed not to watch these two videos more than twice during this phase. After watching these videos, they had to do an initial programming practice exercise. This exercise was used as warm-up exercise and it aimed to assure that they could (i) use Greenfoot, (ii) record their computers' screens while they were doing the exercise and, after finishing it, (iii) upload the resulting video in a given Dropbox account.

The second session was performed at the participants' school and lasted 1.5 hour. It consisted of the last three phases of the study; learning, programming and post-programming phases, as showed in Figure 5. To this session, students were randomly divided into two groups, SE group and SS group. To do the activities in parallel, the groups were placed in different classrooms and each student used their own computer and headphone. To control the activities, each group had a different mediator; the SE group was conducted by the first author of this paper; the SS group was conducted by another researcher from our research group at <omitted for submission> trained for the mediation by the first author.

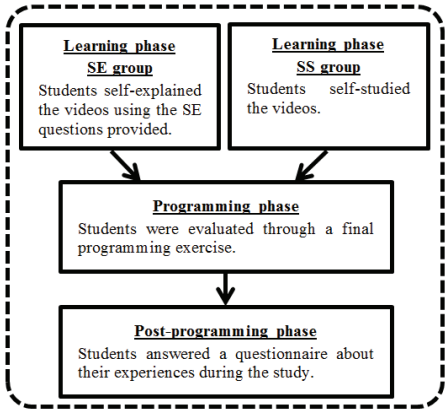

Figure 5. Classroom session.

During the learning phase, students had to watch a set of 4 sections of the episode 4 of The Joy of Code videos. Each video section consisted of a sequence of extension and refinement activities to teach the students how to make an object act in the Greenfoot world. To make the boundaries between two different sequences performed by the teacher, these videos were cut after every refinement activity. After watching each video section, students from both groups had the same amount of time to study it. To collect data about how students in the SE group self-explained each video section using the corresponding SE prompts, students from the SE group were instructed to study each video section by answering the proposed questions on paper. Similarly, students from the SS group had blank sheets of paper if they wanted to write some notes while studying each video section. During this time of study, students from both groups could watch the video section one more time if they wanted to.

To compare the performance of both groups, during the programming phase students were evaluated through a final programming practice exercise. The time for doing this final exercise and the practice exercise itself were identical for both groups. To analyze the students' dynamic process of programming, they were instructed to record their computers screens while they were answering the final exercise and, after finishing it, to upload the resulting video in the Dropbox account they had used previously.

Finally, at the end of this classroom session, students from both groups answered a final questionnaire to evaluate their experience of learning programming from videos in this study. In addition, the SE group evaluated the proposed approach of self-explaining these videos while learning programming.

VI. RESULTS

The results we obtained from the data collected during the online and classroom sessions are presented in the following subsections.

A. Data collected during online session

From the initial questionnaire, we have collected the following data. The SE group had 7 participants, but one of

them had problems with his computer and his data was not used. The remaining SE group students were between the ages of 16 and 19 years old (mean = 17.33). They had no prior programming experience. All of them, but one have used videos from Youtube for learning some subject (e.g., they used videos for learning how to use a software (n=3) and learning how to make a game (n=3)). To study the mentioned subjects, they have used videos in the ways depicted in Table III.

TABLE III. HOW PARTICIPANTS IN SE AND SS GROUPS HAVE USED VIDEOS WHILE STUDYING SOME SUBJECT.

	#students of	
How have you used these videos?	**SE group**	**SS group**
I have paused, forwarded or rewinded in order to pay attention in specific parts of the video	5	3
I have watched more than one time the same video	4	3
I have watched more than one time parts of the same video	2	1
I have tried to reproduce what I was seeing in the video	2	4
I have made notes while I was watching the video	1	1
I have answered questions while I was watching the videos	1	1
I have simply watched the video without making any other activity	1	1

The SS group also had 7 students, but one of them had prior programming experience and his data was not used. The remaining SS group students were between the ages of 16 and 18 years old (mean = 17.17) and they had no prior programming experience. Similar to SE group, all 6 students in SS group, but one have used videos from Youtube for learning some subject (e.g., they used videos for learning how to make games (n=4)). To study the mentioned subjects, they have used videos in the way depicted in the Table III.

The videos of students' computer screens, recorded during the online session showed that all of them could (i) use Greenfoot and the software to record their computers' screens after installing them; (ii) use the Dropbox folder to upload the resultant videos; and (iii) finish the initial programming practice exercise. Then, all of them could use the tools needed for the classroom session of the study.

B. Data collected during classroom session

To answer [RQ1], we examined written answers and videos of students' final programming practice exercise from both groups. These data complement each other, so we looked at them together to grade students' practice exercises. The final practice exercise had 5 questions and students' grades could be a maximum of 40 points. The score in the SE group ranged from 25 to 37 points, with the median at 34.5 and a mean of 33.33 (SD = 4.32). The score in the SS group ranged from 29 to 35 points, with the median at 32 and a mean of 32 (SD = 2.76). However, we cannot confirm that the difference between scores of SE group and scores of the SS group is significant (Mann Whitney's U=11.5, z=-1.052; p=0.29) regarding the final programming practice exercise. Consequently, our data does not enable us to distinguish between the approaches used in the SE group and in the SS group respectively.

To answer [RQ2], we examined students' SE written answers together with the scores of SE group students' final programming practice exercise. Each answer to the SE questions was graded in a maximum of 1 point (wrong answer – 0; half-right answer – 0.5; correct answer – 1). We proposed a set of 38 SE questions, so in total, students could have a maximum score of 38 points. The scores for the SE answers ranged from 16 to 26.5 points, with the median at 22 and a mean of 21.36 (SD = 3.76). We found a significant correlation (Spearman's rho=0.899, p=0.007) between the scores obtained from students' SE answers during learning phase and the scores of students' final programming practice exercises answers during programming phase.

At the end of the study, we asked for students' feedback through a final questionnaire. When asked about their experience studying from videos, SE group presented positive feedback. Students found it easy (n=4), because they could re-watch the videos in case of doubts (n=2) or reproduce the content they learned from the videos using Greenfoot (n=1). Other adjectives used were: interesting, great and fun. When asked about using videos combined with SE questions, SE group also presented positive feedback. They found it helpful (n=4), because the questions helped them to remember things (n=2), the questions were important to summarize the information learned (n=1), and because they could monitor what they had learned (n=1). Despite this positive feedback, our approach of videos combined with SE questions was considered very time-consuming (n=4).

Similarly, SS group also presented positive feedback when asked about their experience. Students found to study from videos was easy to understand the subject (n=4). In particular, they said The Joy of Code videos were very good, simple and more understandable than others they had watched previously because the teacher presented the code as a simple step-by-step guide and at an appropriate speed. When asked about the activities they performed while studying from videos, they said they wrote down on paper what they found important and hard to remember (n=2).

C. Discussion

Examining the scores from students' final practice exercises in SE group and SS group, we could not confirm the difference between the scores from the two groups was significant. Therefore, the SE effect could not be replicated in this study because of a combination of two reasons. First, the content learned might have been too simple for the participants. In general, studies that support the SE effect in programming only looked at more complex content than those we have investigated. However, due to time constraints, students had to learn very simple tasks during its learning phase. Consequently, students in both groups did not experience a high level of difficulty in studying and solving associated problems in the present study. This issue was confirmed with students' feedback about learning from videos.

Second, the instructional explanations presented in the videos also might have contributed for this result. These instructional explanations contained the answers to the SE questions that the instructional group had to answer. Besides that, we observed from SS group's notes that some of them

were equivalent to answers to SE questions. For example, notes from five students in SS group were related to the code produced. These notes correspond to answers to the questions related to a refinement activity in SE group. Thus, it seems that students in SS group benefited from instructional explanations presented in the videos and, in consequence, engaged in a learning activity that was similar to that of SE group. As a result, students in SS group had similar learning outcomes when compared to the students in SE group.

Besides that, we found a significant positive correlation between scores for the SE answers and scores of students' final programming practice exercise in the SE group. Our analysis showed that almost 90% of the variation in the scores of SE group final practice exercise can be explained by the variation in the scores for their answers to SE questions. Thus, students from SE group who answered more correctly SE questions in learning phase had the best performance in programming phase.

Besides that, students had a positive feedback regarding our approach. Regarding the use of videos as instructional materials in this study, students' opinions from both groups suggested they enjoyed it. Regarding the use of SE questions, most of the students in the SE group found it helpful. Although students of SE group had a good experience using our approach, most of them considered the approach very time-consuming.

VII. CONCLUSION AND FUTURE WORK

In this article, we presented our approach to learning programming through stepwise self-explanations of examples presented in videos. To evaluate our approach, we carried out an experimental study with second-year students in a technical high school in Denmark. We assumed that by learning programming through our approach students in SE group would outperform students in SS group. However, we could not confirm that the difference between the two groups was significant, and, as a consequence, we could not replicate the SE effect in the context of this study. In our opinion, the lack of effect happened because of these reasons: (i) the low level of difficulty of the content chosen and (ii) the instructional explanations presented in the videos.

Despite this result, the significant correlation between learning and programming phases within SE group suggests the proposed SE questions served as a useful starting point in our approach. This result also suggests our approach should be investigated in further detail, especially with regard to which instructional conditions are more effective for it. Because of that, as future work, we should (i) choose a more difficult content in the programming domain to carry out a next experiment and (ii) be concerned not only about how to phrase SE questions but also about the most appropriate time to ask those SE questions while students are watching videos.

REFERENCES

[1] Robins, A.; Rountree, J.; Rountree, N. Learning and Teaching Programming: A Review and Discussion, Computer Science Education, v. 13, n. 2, p. 137-172, 2003.

[2] Chi, M. T. H.; Bassok, M.; Lewis, M.; Reimann, M., & Glaser, R. Self-explanations: How students study and use examples in learning to solve problems. Cognitive Science, 13, 145-182, 1989.

[3] Chi, M. T. H; Deleeuw, N.; Chiu, M.; Lavancher, C. Eliciting self-explanations improves understanding. Cognitive Science, 18, 439-477, 1994.

[4] Bielaczyc, K.; Pirolli, P.; Brown, A. L. Training in self-explanation and self-regulation strategies: Investigating the effects of knowledge acquisition activities on problem solving. Cognition and Instruction, 13, 221-253, 1995.

[5] Bennedsen, J.; Carpersen, M. E. Exposing the Programming Process. In Reflection on the Theory of Programming: Methods and Implementation, J. Bennedsen, M. E. Carpersen, M. Kölling (Eds.) Springer-Verlag, 6-16, 2008.

[6] Caspersen, M. E. Educating Novices in the Skills of Programming. PhD dissertation PD-07-4, Department of Computer Science, University of Aarhus, 2007.

[7] Caspersen, M. E.; Kölling, M. STREAM: A First Programming Process. Trans. Comput. Educ., v. 9, n. 1, p. 1-29, 2009.

[8] Kölling, M. The Joy of code [Video recordings]. 2012. Available: http://bit.ly/1jbTbHq

[9] Biggs, J.; Tang, C. Teaching for Quality Learning at University. New York, N.Y.: McGraw-Hill/Society for Research into Higher Education/Open University Press, 2011.

[10] Chiu, J. L.; Chi, M. T. H. Supporting self-explanation in the classroom. In V.A. Benassi, C.E. Overson, & C.M. Hakala (Eds.). Applying science of learning in education: Infusing psychological science into the curriculum, 2014. Available: http://bit.ly/KbYLtG

[11] Clark, R. C.; Nguyen, F.; Sweller, J. Efficiency in Learning: Evidence-Based Guidelines to Manage Cognitive Load. Wiley, 2005. ISBN 978-0-7879-7728-3.

[12] Berthold, K.; Renkl, A. Fostering the understanding of multi-representational examples by self-explanation prompts. In: BARA, B. G.;BARSALOU, L., et al, Proceedings of the CogSci 2005, 2005. Erlbaum. p.250-255.

[13] Pirolli, P.; Recker, M. Learning Strategies and Transfer in the Domain of Programming. Cognition and Instruction, v. 12, n. 3, p. 235-275, 1994/09/01, 1994.

[14] Renkl, A. Learning from Worked-Out Examples: A Study on Individual Differences. Cognitive Science, v.21, n.1, p.1-29, 1997.

[15] Crippen, K. J.; Earl, B. L. The impact of web-based worked examples and self-explanation on performance, problem solving, and self-efficacy. Comput. Educ., v. 49, n. 3, p. 809-821, 2007.

[16] Kwon, K.; Kumalasari, C. D.; Howland, J. L. Self-Explanation Prompts on Problem-Solving Performance in an Interactive Learning Environment. Journal of Interactive Online Learning, v. 10, n. 2, p. 96-112, 2011.

[17] Yen-Chu, H. Combining Self-Explaining With Computer Architecture Diagrams to Enhance the Learning of Assembly Language Programming. Education, IEEE Transactions on, v. 55, n. 4, p. 546-551, 2012.

[18] Lee, M. J.; Ko, A. J.; Kwan, I. In-game assessments increase novice programmers' engagement and level completion speed. Proceedings of the ninth annual international ACM conference on International computing education research. San Diego, San California, USA: ACM: p. 153-160, 2013.

[19] Vihavainen, A.; Miller, C. S.; Settle, A. Benefits of Self-explanation in Introductory Programming. Proceedings of the 46th ACM Technical Symposium on Computer Science Education. Kansas City, Missouri, USA: ACM: p. 284-289, 2015.

[20] Kölling, M. Introduction to Programming with Greenfoot: Object-oriented Programming in Java with Games and Simulations. Prentice Hall, 2010. ISBN 9780136037538.

[21] Gomes, A., Mendes, A.J. Learning to program - difficulties and solutions. International Conference on Engineering Education – ICEE 2007, Coimbra, Portugal.

[22] Prensky, M. R. Teaching Digital Natives: Partnering for Real Learning. SAGE Publications, 2010. ISBN 9781452271408.