Exploring Design Characteristics of Worked Examples to Support Programming and Algorithm Design

Camilo Vieira
Purdue University
401 N Grant St Office 372,
West Lafayette, IN 47907
+1(765)250-1271
cvieira@purdue.edu

Junchao Yan
Purdue University
401 N Grant St Office 372,
West Lafayette, IN 47907
+1(765)337-8867
yan114@purdue.edu

Alejandra J. Magana
Purdue University
401 N Grant St Office 256,
West Lafayette, IN 47907
+1(765)494-3994
admagana@purdue.edu

ABSTRACT
In this paper we present an iterative research process to integrate worked examples for introductory programming learning activities. Learning how to program involves many cognitive processes that may result in a high cognitive load. The use of worked examples has been described as a relevant approach to reduce student cognitive load in complex tasks. Learning materials were designed based on instructional principles of worked examples and were used for a freshman programming course. Moreover, the learning materials were refined after each iteration based on student feedback. The results showed that novice students benefited more than experienced students when exposed to the worked examples. In addition, encouraging students to carry out an elaborated self-explanation of their coded solutions may be a relevant learning strategy when implementing worked examples pedagogy.

Categories and Subject Descriptors
K.3.2 [**Computers And Education**]: Computer and Information Science Education – *Computer science*

General Terms
Algorithms, Human Factors.

Keywords
Computational Thinking, Programming Education, Worked Examples.

1. INTRODUCTION
Computational thinking [24] has emerged as a set of concepts and skills that enable people to understand and create tools to solve complex problems [20]. Many programming and algorithm design processes have been proposed as part of this set of understandings and skills [7, 8]. Hence, it is relevant to introduce programming and algorithm design as part of undergraduate courses; however, learning to program is a complex task [19]. Thus, it is necessary to explore scaffolding strategies to introduce these computational thinking skills. The use of worked examples has been demonstrated to be an effective approach for supporting complex learning when it is guided under certain principles [5]. It can reduce the extraneous cognitive load, which is not beneficial to learning. Therefore, it allows the learner to devote cognitive resources to useful loads.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Copyright ©JOCSE, a supported publication of the Shodor Education Foundation Inc.

This study explores how worked examples can be paired with programming and algorithm design. The guiding research questions are:

- How can worked examples be effectively designed to introduce programming concepts to novice learners?
- How do students self-explain worked examples when approaching a solution to a programming assignment?

2. BACKGROUND
Learning how to program is a difficult task [19]. Programming courses are considered the most challenging at the undergraduate level as they often have the highest dropout rates. In order to learn to program, a student has to understand (a) the purpose of a program, (b) how the computer executes programs, (c) syntax and semantics of the programing language, (d) program structure, and (e) how to actually build a program [9]. Since the learning process involves many steps, these myriad steps may generate a high cognitive load for students who have no previous experience in algorithm design or programming languages.

Researchers have identified differences in the way novices and experts experience programming tasks. Experts use specialized schemas to understand a problem based on its structural characteristics [19]. They use problem solving strategies, such as decomposing the program and identifying patterns, in order to approach a solution [18]. Language syntax and analyzing line-by-line details of programs tend to be the focus of novices due to the superficiality of these skills in the hierarchy of knowledge. [18]. They usually have problems related to language constructs, such as variables, loops, arrays, and recursion.

The use of worked examples (WE) has been recognized as a relevant strategy for supporting novices in learning tasks that involve a high cognitive load. Worked examples approach is guided by principles associated with Cognitive Load Theory (CLT). CLT is a recognized theory that focuses on cognitive load processes and instructional design [15]. CLT establishes a cognitive architecture to understand how learning occurs. The cognitive architecture structures memory that comprises a limited working memory and a vast long-term memory [10]. CLT states

that there is a cognitive load generated when learning occurs. This load can be affected by the learner, the learning task, or the relation between the learner and the learning task [10].

There are different types of cognitive loads: (1) intrinsic load, which is the inherent load to the difficulty of the learning task; (2) germane load, which is comprised of required resources in the working memory to manage the intrinsic load, which, in turn, support learning; and (3) extraneous load, which refers to the load that arises from the instructional design and does not directly support learning [17]. To improve the learning process, the extraneous load should be minimized so that the germane resources can be maximized.

Figure 1 shows a representation of the learning process based on the CLT. In Phase 1, a student is working on a task that has different forms of representations. In Phase 2, the cognitive process that takes place is depicted. In Phase 2a, different senses process the information the student is receiving. In Phase 2b, the limited working memory (three to five chunks of information) is assigned to germane or extraneous resources depending on how the load arises from the learning experience. In Phase 2c, schemas are created and stored in the long-term memory when learning takes place.

Novices usually work backward to solve problems using a means-ends analysis. Students need to fill the gap between the initial problem state and the final goal, and this search process generates a cognitive load. Some of the instructional design techniques that have been proposed to reduce the cognitive load are: goal free effect, worked examples effect, and completion effect [10]. The goal free effect suggests that, by removing a specific goal from the problem, the learners will work forward, state by state as experts do. Thus, the cognitive load is reduced because the students only have to consider the current state and the next state of the problem. The worked example effect occurs when students are exposed to an expert solution to the problem. These examples allow learners to start solving similar problems by analogy, thereby reducing the cognitive load. Finally, the completion effect refers to problems with a given partial solution that are provided to students to complete. Completion effect examples are gradually modified to present less information to the student. This approach is also called faded worked examples (FWE), and it has shown positive results in reducing cognitive loads in the domains of mathematics and programming.

Atkinson et al. [5] proposed instructional principles for the design of WE based on several studies. According to these principles, a WE should include: (1) A problem statement; (2) A procedure for solving the problem; and (3) Auxiliary representations of a given problem. Atkinson's principles and their adaptations for this study are summarized in Table 1.

When WE are used as part of the learning process, the student goes through a four-stage process described by the theory of Adaptive Control of Thought-Rational (ACT-R) [4]. According to this theory, the skill acquisition process is composed of four stages in which knowledge transitions from declarative to procedural [5]. During the first stage, the students solve problems by analogy using worked examples. Then, in the second stage, the students use abstract declarative rules gathered from the examples. When students get to the third stage, declarative knowledge is already acquired and stored in their long-term memory. Procedural rules have also started to become clearer to students by practice. Therefore, students are able to respond automatically and faster to familiar problems. During the last stage, once students have been exposed to several examples, they are able to solve many different problems on their own.

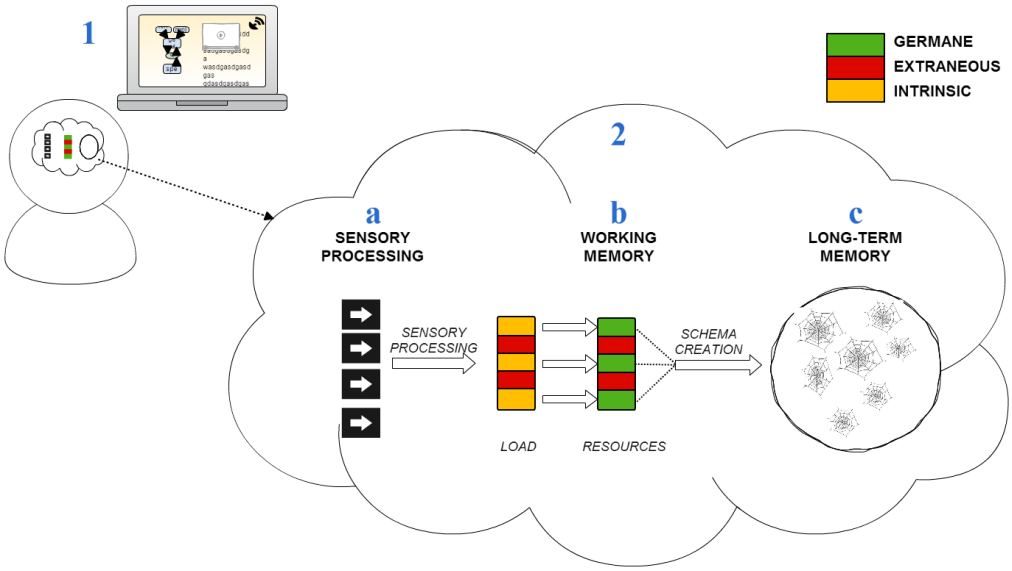

Figure 1. Learning process from a CLT perspective. In (1) the learner is studying the materials. In (2) the learning process takes place: (a) Senses capture information; (b) Different forms of cognitive load make use of working memory; (c) Schemas are created and automatized in long-term memory

Table 1. Effective features of worked examples described by Atkinson and collaborators [5]. The right column describes our adaptation for these learning tasks

Feature	Description	Our Adaptation
Intra-Example	• The use of multiple formats and resources is important when designing WE; however, different formats should be fully integrated to avoid extra cognitive load generated by the split attention effect. • The example should be divided in sub goals or steps to make it easier for the student to understand. Labels and visual separation of steps can be used for this purpose.	• The examples contained multiple forms of representations including C# code, visual flowchart algorithm description, and verbal explanations of the approach. • Each representation was segmented in steps toward the solution. These steps were aligned with the representations.
Inter-example	• The variability of problems during a lesson can offer learning benefits, but it is important to reduce the cognitive load when using techniques such as WE. • The use of multiple WE (at least two examples) with structural differences can improve the learning experience. The WE should be presented with similar problem statements that encourage the students to build schemas based on analogies and the identification of declarative and procedural rules.	• Different problems were approached during each lab session. • Two examples were provided and students were required to complete at least three additional programming challenges. • All the examples and challenges were focused on a specific topic for each lab session (e.g. loops, creating arrays, searching in arrays).
Environmental	• Students should be encouraged to self-explain the WE in order to be actively engaged with them. • Some strategies that support this process are: (1) Labelling WE and using incomplete WE; (2) Training Self-Explanations; and (3) Cooperative Learning.	• One of the examples did not have the verbal explanation (i.e. in-line comments of the programming code). • Before starting to solve the assignment, students were asked to comment on the code for the example that did not have verbal explanation. This activity was intended to encourage self-explanation of the examples. • Some of the assignments could be built from the examples.

2.1 Previous Experiences

Guzdial [11] has advocated for an approach to programming education other than a common approach, which asks students to just start building a program. Based on Kirschner, Sweller and Clark [12], he argued that, "expecting students to program as a way of learning programming is an ineffective way to teach" (p.11). As an alternative, he proposed an approach based on the work of Pirolli and Recker [16] who used WE and cognitive load theory to introduce programming concepts. In one of their experiments, Pirolli and Recker explored how transfer occurs in learners, starting with examples and moving on to programming problems on Lisp. To implement the examples, each lesson started by having students read the textbook and analyze WE. The students then used this knowledge to find a solution for an assigned problem. Authors hypothesized that the problem solving process enriched declarative knowledge as well as procedural knowledge.

The declarative knowledge in programming includes code structure, programming abstractions, functionality of the abstractions, and purposes and operation of the program. All these elements are represented as a mental model.

On the other hand, the procedural knowledge comprises the construction, manipulation, and interpretation of this model. In their experiments, Pirolli and Recker [16] found that worked examples were useful in building these mental models by "providing [students with] concrete referents for abstract discourse and newly introduced concepts and propositions" (p.273).

In another study, Moura [13] used Portugol, a tool for learning algorithms, for students to understand a given example by visualizing the execution of the algorithm. She found that, although students took some time to get used to the tool, once they did get used to it, they performed better on assessment tests when learning computing science fundamentals. Regarding the implications of the study, Moura suggested that an effective way to help students learn how to program requires an easy-to-use tool as well as assigning some pre-training time for the students to get familiar with it.

This study focuses on a strategy for providing worked examples to an introductory programming course to support student learning of process of loops and arrays concepts. The worked examples were designed following the principles by Atkinson and colleagues [5] as described on Table 1.

3. METHODS

This study followed a Concurrent Mixed Methods Research process design [23]. This design includes one quantitative strand (pretest, posttest, survey, and lab scores) and one qualitative strand (open ended questions and comments in the code of the examples). Each strand was analyzed independently. At the end, the identified commenting styles were related with the quantitative measures to evaluate whether there was a trend in the way students experienced the use of examples.

3.1 Participants

The participants of this study included thirty-five undergraduate students majoring in Computer and Information

Technology at a large midwest university. As part of an introductory programming course, they were exposed to weekly lab sessions where they applied programming concepts learned in lecture. Three of these weekly sessions (8th, 9th, and 10th) were used to evaluate the WE approach. The sample size as well as the participants slightly varied from session to session since not all students attended all the sessions or completed pretest and posttest assessments.

These students were divided into two different groups. For lab session #8, both groups used worked examples. For lab sessions #9 and #10, one group was considered the experimental group (using WE) while the other one was the control group. The control group continued doing the lab session as they were used to; that is, solving the assigned problems based on what was learned during the lectures without additional scaffolding but only the help provided by the teaching assistant. Table 2 summarizes participants' information and configurations for each of the sessions.

Table 2. Number of participants per session

Session	Group	Number of Participants	Programming Experience
8th	Experimental	28	12
	Total	28	12
9th	Experimental	19	10
	Control	15	7
	Total	34	17
10th	Control	16	9
	Experimental	14	8
	Total	31	17

3.1 Materials

Two examples designed by following the instructional principles of worked examples [5] were provided to the students in the experimental group. The examples were composed of a Visual Studio Solution with the programmed worked examples as well as a matched flowchart representing the solution. Figure 2 depicts an example of what was provided to the students. On the left side, the C# code adds the even numbers from 0 to a variable n1. The code included comments to explain each section. The right side of the figure showed a flowchart describing the algorithm design for this particular implementation. This design was coupled with textual description. All the elements in the example were identified by a code that allowed the students to match the different representations (i.e. steps in the flowchart with segments of code).

Two examples were provided to the students per session. Both had the distribution depicted on figure 2; however, the comments within the code were not included in the second example because the students were required to complete that portion as part of the assignment. With this design, we expected that students would start using the examples to solve the problems by analogy. Then, having acquired some declarative and procedural rules, students were expected to be able to solve the different and more challenging problems on their own.

3.2 Procedures

The research protocol consisted of a series of tasks. The first task was a pretest aligned to the learning objectives of the lab session. The students were given 10 minutes to complete the test before starting the session. The next tasks consisted of exploring examples and commenting on the code in one of the examples in order to self-explain it. The fourth task was to solve three additional assignments using the examples whenever they were necessary. Finally, students completed a posttest and survey related to their perceptions regarding the use of worked examples. The students could take as much time as they needed to complete this task within a period of two hours. The assignments had to be turned in before proceeding to the next assignment. The collected data included the pretest, posttest, survey data, commented example, and programming projects.

Following design-based research approaches [21], this study includes three iterations, one for each lab session. Right after the session, the tests and survey were analyzed. This information was used to refine the examples, assignments, and instruments for the subsequent iteration. For example, after the first iteration, some students mentioned that the comments in the code were very detailed decreasing the code's readability. Therefore, the next iteration examples included simpler comments.

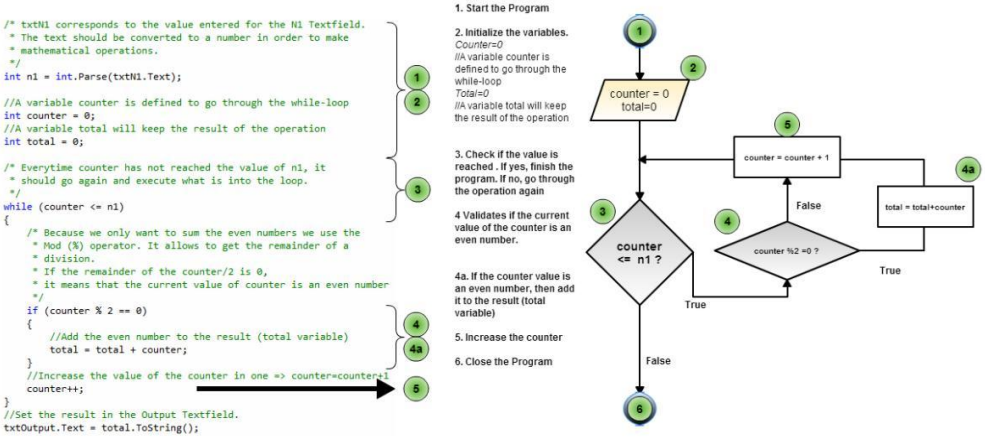

Figure 2. Distribution of a worked example including multiple representations of the solution (i.e. computational, textual, and graphical)

3.3 Data Collection

3.3.1 Learning

Pretest and posttest as well as lab assignments were employed to assess learning gains during the lab sessions. Table 3 summarizes how these instruments were prepared and implemented. The exercises for the pretest and posttest were slightly modified (i.e. changing values or sizes) to minimize the testing effect. Written comments within the code were also used as an additional source of qualitative data. Students were required to do this as a strategy to self-explain one of the provided examples.

Table 3. Description of the learning instruments employed in each laboratory session

Session	Description	Pretest and Posttest	Lab Assignment

8th	The student will create an algorithm that contains for-loop and while-loop structures to solve summations or display a list of values	Four exercises to calculate the output after certain loop structure. For example:
		`int count = 1;`
		`int sum = 0;`
		`while (count <= 10)`
		`{`
		`  sum = sum + 5;`
		`  count++;`
		`}`
		`MessageBox.Show(sum.ToString());`
			Complete the implementation for the following list of functions:
			(1) Sum of evens between 0 and a given value N
			(2) Sum of a range of numbers
			(3) Calculate the factorial of a given value N
			(4) Calculate $1 + 2^2 + 3^2 + \ldots N^2$

9th	The student will create an algorithm that initializes an array, add some values, and display the stored values	Four exercises to complete the code or write the output of certain algorithm. For example:
		`_____ mNumbers = _____;`
		`int k;`
		`int m = 1;`
		`for(k = 0;  k <= _____; k++)`
		`{`
		`  _____;`
		`  _____;`
		`}`
			Write the code to store and display numerical and textual values in an array.

10th	The student will create an algorithm to perform a sequential search and switch array elements	Four exercises to write/complete the code to find an element within an array, reverse an array or switch two values within an array. For example:
		`int [] mNum = new int[5] {3, 9, 2, 7, 6};`
		`int position = 0;`
		`for (k = 0; _____; k++)`
		`{`
		`  if (_____)`
		`  {`
		`  _____`
		`  }`
		`}`
			Write the code to complete the following methods for a given array:
			(1) Add
			(2) Find
			(3) Switch
			(4) Merge
			(5) Reverse

3.3.2 Perceptions

At the end of the lab sessions, students were given a survey where their responses were recorded using a seven-level Likert Scale with scores ranging from 0 to 6 from strongly disagree to strongly agree. The results were normalized from 0% to 100%. Values between 0% and 40% were considered negative perceptions. Values from 40% to 70% were considered undecided perceptions and values higher than 70% were considered positive perceptions.

The following questions were asked in the survey: (1) I feel I have the ability to accurately evaluate and construct a <concept>; (2) I feel I have the ability to describe a <concept>; (3) I have the ability to create a program that includes a <concept>. The three questions were posed to assess perceived ability to complete the given tasks. Two additional open-ended questions were asked to students to analyze their perceptions about the examples and the laboratory session: (1) What would you improve for the examples; and (2) What suggestions do you have for the laboratory sessions?

3.4 Data Scoring and Analysis

3.4.1 Learning

Pretest and posttest assessments were scored by two different graders to assure reliability. Whenever the graders got different scores, they discussed the scores until they agreed on a certain value. The lab assignments were scored by the teaching assistant. The comments written by the students were analyzed qualitatively to identify different categories in which the comments could fit. These categories were assigned a descriptive code that was used later to identify students' commenting styles.

3.4.2 Perceptions

Descriptive and inferential statistics were used to analyze the learning and perception measures. Whenever the data did not satisfy the normality assumption, a logarithmic transformation was used to be able to run the inferential tests. The open-ended questions were first analyzed using open-coding by one of the researchers. Then, to assure reliability, another researcher re-analyzed students' responses using his codes. The percentage of agreement was 80%. These codes were then grouped by themes.

4. RESULTS

Three iterations of data collection are reported in this section. At the end of each iteration, quantitative and qualitative results were used to improve the learning materials and the instruments for the following iteration.

4.1 Session #8: The student will create an algorithm that contains for-loop and while-loop structures to solve summations or display a list of values

4.1.1 Quantitative Data

Pretest and posttest scores were compared to evaluate learning gains. No significant differences were found for the complete group of students t (54) = -0.702, p = 0.4857 nor for the subgroups (i.e., students with/without programming experience). Table 4 depicts the descriptive statistics for the learning measures from session #8.

Differences between groups were assessed by comparing lab score and time to complete the assignment. Significant differences were only found in the "time to complete" variable, and these differences were found when comparing students who had programming experience and those who had not $F(26,1)=23.86$, $p<0.001$. however, although non-significant differences were found, students without programming experience increased their score from pretest to posttest more than those with some experience. They also received a higher lab score as compared to students with prior programming experience.

Overall, perception measures fell in the positive perception category for the ability construct (Mean = 79.49%; Standard Deviation –SD– = 16.61%). The measure was also compared between groups. Significant differences were found for the ability construct ($t(24)=3.204$, $p<0.01$) when compared by programming experience. The results suggest that students with previous programming experience (Mean = 89.90%; SD = 15.48%) perceived a higher ability to deal with loops than those without previous experience (Mean = 71.85%; SD = 13.19%).

Table 4. Descriptive statistics of student learning scores in Lab Session #8

Test		Overall (N=28)	Programming Experience	
			Yes (N=12)	No (N=16)
Pretest (%)	Mean	50.71	64.58	40.31
	SD	30.81	24.90	31.38
Posttest (%)	Mean	56.25	65.42	49.38
	SD	28.14	29.81	25.62
Lab Score (%)	Mean	95.71	95	96.25
	SD	8.36	9.05	8.06
Time to Complete (min)	Mean	86.32	68.83	99.44
	SD	22.29	17.66	15.42

4.1.2 Qualitative Data

The two open-ended questions were completed by twenty-five students. The questions were: (1) What would you improve for the examples?; and (2) What suggestions do you have for the laboratory sessions? Table 5 and Table 6 depict the results of the qualitative analysis to students' responses. A group of students suggested getting rid of some of the comments (24%) or better aligning the examples with the assignments (16%).

Table 5. Categorical analysis for the student responses to the strategies to improve examples in Lab Session #8

Theme	Code	Definition	%	Representative Quote
Students struggled with specific elements within the examples	Nothing to Improve	The student thinks that the examples are fine the way they are presented	56	"*nothing to improve*" / *"Nothing I can think of. As long as they are related to the problems and the comments are descriptive, they are fine"*
	Less Comments	The student highlights the need to get rid of some of the comments since they have an impact on the code readability	24	*"Less comments, too hard to find place among a sea of comments"/ "I feel like the comments chitter the code and makes it difficult to read"*
	Math Expression	The student feels the use of unknown mathematical expression constrains her/his understanding of the example	16	*"Explain N! - What that means?"* / *"…the '^' syntax issue was confusing for me"*
	Explicit relation example / assignment	The student requests that the examples be more detailed so that they guide the student through the problem solving process of the assignment	16	*"Better descriptions for what we are supposed to do" / "Make it so the examples demonstrate most of the common types of loops people mess up on."*
Students suggested integrating more hands-on activities as part of the classroom approach	Better with Examples	The student thinks working with examples is a better approach than working from scratch	20	"*I wanna spend more time with examples*" / "*It helped a lot but I feel like the book could've helped explain writing the math problems more in depth*"
	In-class activities	The student thinks that the class activities should be focused on practical activities (design and programming activities)	12	"*Maybe more hands on in class and allow us to program it on the computers"* / *"Make students answer questions in algorithmic form*"
	Better without Examples	The student does not consider the examples as having helped her/him to solve the assignment	4	*"Unsure, did not use them"*

Table 6. Categorical analysis for the student responses to the strategies to improve the laboratory sessions in Lab Session #8

Theme	Code	Definition	%	Representative Quote
Students' suggestions about laboratory sessions	No suggestion	The student thinks the laboratory sessions are fine the way they are carried out	64	*"They are going good" / "Nothing so far"*
	Logistic Improvements	The student feels that the laboratory session could be improved by either having more time, different levels of difficulty, or more teaching assistants	32	*"More TAs for more help" / "More optional assignments" / "More TA to speed things up."*
	Exploring examples	The student thinks that exploring examples would help them to better understand the concepts before starting to build a program from scratch	20	*"To continue to experiment with these types of ideas on presenting programming in an easier to understand format" / "Explore and try various examples".*
	Better without examples	The student does not consider the examples as having helped her/him to solve the assignment	12	*"I personally like the old method better" and "Keep them from Scratch".*

Table 7. Categorical analysis for the student comments within the second example in Lab Session #8

Category	Definition	%	Representative Quote
1. Detailed Comments	The student wrote a detailed description in every step of the code	21	**radSumOfNumbers is the name of the Radiobutton* **related to the sum of numbers* **it allows to identify whether the user wants to perform* **this operation when checked or when the radio button is not checked. */* */*txtN1.Text is what the user enters* ** The text should be converted into a number froms string to do mathematical operations */* *//txtN2.Text will correspond to n2, because that is what the user enters* *// the parse method converts the string into numbers* *in the for loop here, the variable i is defined as n1,* ** i <= n2, will make sure the loop will continue until the number reaches the* ** the number the user entered for n2* ** i++ will make sure the count will increase 1 in every loop.* *//Add the sum to the total result* *//shows the result in the output textbox*
2. Basic Comments	The student used the first example to write the comments for the second one. The comments were very simple.	32	*"*radSumofNumber is the name of the radio button* **if this radio button is checked, the loop/calculation are executed* *//declare N1 and parse* *//declare N2 and parse* *//declare the initial value for total* *//create the loop with the variables* *//calculation from the loop* *//display the calculation*
3. No Clear Comments	The student did not write any comments at all or the comments were too incomplete to be understood.	18	*// adds together all the inputed values*
4. Relevant Conditions	The student only focused on relevant sections of the code (e.g. loop conditions) with rich descriptions.	29	*// * the total is initialized to zero* *// * i equals n1 in the beginning of the code then as long as i is smaller than n2 than* *// the program will operate and it will add 1 to n1 after every time.* *// i is added to the total every time that the program is run.* *// The output is displayed through by using the tostring method.*

Regarding the laboratory sessions, students' perceptions were divided between those who preferred working with examples (20%) and those who preferred solving problems from scratch (12%). The other source of qualitative data was students' comments in the code for one of the provided examples. Four categories were identified for the commenting styles from students. The categories, descriptions, and examples are presented in Table 7. Most of the students either used the first example as a model to comment the other one with simple comments, or focused on describing the most relevant section of the code.

4.1.3 Quan + Qual

In addition to the qualitative analysis of the comments, we wanted to evaluate if there was a quantitative difference among students with different commenting styles. Table 8 shows descriptive statistics for the learning and perception measures grouped by commenting style.

Non-significant differences were found from pretest to posttest for all of these groups; however, the highest scores in both posttest and lab scores were from students with either detailed comments or those who highlighted relevant conditions with their comments. These students also spent more time completing the assignment on average compared to the rest of the students. We speculate that these non-significant difference may be due to a large standard deviation and the small sample size, which resulted from dividing the students into four groups.

Significant differences were found for the Ability Construct between the commenting styles "Basic" and "Unclear." The results suggest that students who did not write comments or who wrote unclear comments felt very confident in their abilities. On the other hand, those with basic comments may have felt unsure of their abilities; therefore, their comments were as simple as possible.

Table 8. Descriptive statistics of learning and perception scores grouped by commenting styles in Lab Session #8

Commenting Style	Pretest (%)		Posttest (%)		Lab Score (%)		Time to Complete (min)		Ability (%)	
	Mean	SD	Mean	Mean	Mean	SD	Mean	SD	Mean	SD
1. Detailed (N=5)	58.33	37.64	75	27.39	96.67	8.16	91.83	26.44	82.22	14.91
2. Basic (N=9)	38.33	33.16	45.55	30.46	95.56	8.81	77.33	18.49	67.90	18.59
3. Unclear (N=5)	55	20.92	49	26.32	92	10.95	77.20	26.37	92.22	10.83
4. Relevant (N=7)	56.25	29.12	58.75	23.87	97.5	7.07	98	16.86	83.33	10.14

4.1.4 Evaluation of the Iteration

As part of the results, two elements were called to our attention from this first iteration: (1) there were no significant differences from pretest to posttest; (2) students requested improvement of the examples by removing detail in the comments but increasing explanations.

After analyzing the results in the pretest and posttest measures, it was identified that some students were able to understand how a loop worked, but they failed to calculate the resulting value that was asked for in the test. Another identified aspect from the test was that students were struggling with mathematical expressions that are common in pseudo-code but might not be that common for them (e.g., "^" to indicate potentiation). Therefore, the following tests were more focused on building/completing code and all the potentially confusing terms were removed. Besides, the comments in the examples were organized in such a way that only the main portion of the code had a rich description of the solution.

4.2 Session #9: The student will create an algorithm that initializes an array, add some values, and display the stored values

4.2.1 Quantitative Data

During this session, the two groups were exposed to different approaches. One of the groups used examples (Experimental, N=18), while the other group used their traditional problem solving approach (Control, N=14). Table 9 shows descriptive statistics for the learning measures of these groups. The programming experience values were only calculated for the experimental group since that is the only group where these may have an impact for assessment.

Non-significant differences were found between groups or between pretest to posttest. In spite of this, it is interesting to see that students without programming experience performed better - and with a smaller standard deviation- in the lab score than students with programming experience. This follows the trend from lab session #8.

Table 9. Descriptive statistics of student learning scores in Lab Session #9

Test		Group		Programming Experience	
		Control (N=14)	Exper. (N=18)	Yes (N=10)	Yes (N=18)
Pretest (%)	Mean	63.09	55.56	59.17	51.04
	SD	29.55	36.04	37.98	35.47
Posttest (%)	Mean	67.86	63.43	70	55.21
	SD	30.29	35.14	33.38	37.78
Lab Score (%)	Mean	79.14	91.67	88.5	95.63
	SD	35.36	23.45	31.27	6.78
Time (min)	Mean	81.57	78.28	76.80	80.13
	SD	26.65	10.70	11.69	9.78

For the ability construct, students in both control group (Mean = 83.33%; SD = 15.71%) and experimental group (Mean = 70.37%; SD = 26.61%) showed a positive perception. Non-significant differences were found between groups. For the experimental group, contrary to lab session #8, differences in ability were not

found between experienced (Mean = 76.11%; SD = 24.71%) and non-experienced (Mean = 63.19%; SD = 28.78%) programmers.

4.2.2 Qualitative Data

At the end of the session, students responded to two open-ended questions: (1) what would you improve for the examples?; and (2) what suggestions do you have for the laboratory sessions? This time only one student suggested that the examples would benefit from having still less comments while another commented: "This was much better without all the comments." In addition, more than sixty-percent of the students thought the examples were complete and useful. Table 10 and Table 11 summarize the results of the qualitative analysis to students' responses.

Regarding the suggestions for the lab session, more than sixty percent of students thought the examples were fine the way they were implemented. As in lab session #8, results suggest that there are differences regarding the preference of using examples. While there is a broad acceptance concerning the way the examples are presented and some of the students really enjoy using this scaffolding, there is another group of students who preferred building their code from scratch.

The self-explanation process of writing comments on the code was only required for the experimental group. The same categories were found for the commenting styles compared to the lab session #8. The distribution of students' comments was: detailed comments (33.33%), basic comments (33.33%), no clear comments (16.17%), and relevant conditions (16.17%).

4.2.3 Quan + Qual

Table 12 shows the results of the comparison of the learning and perception measures grouped by commenting style. The reduced sample size due to the separation between experimental group and control group makes it difficult to use inferential statistics. As in lab session #8, in lab session #9, non-significant differences were found for all of the learning measures of these groups; however, once again, the highest scores were for students who had detailed comments or highlighted relevant conditions.

Non-significant difference was found between the groups for the perception measures. We see, however, that the students without comments or with unclear comments are those who feel more confident about their ability. This result is similar to lab session #8. Students with basic comments present the lowest scores for the perception construct.

4.2.4 Evaluation of the Iteration

For lab session #9, students' suggestions about the examples changed significantly in terms of the number of comments. Still, a couple of students considered the amount of comments could be reduced. Therefore, even simpler but explanatory comments were included in the following example. In addition, students suggested adding more complexity to the examples and programming challenges. Since lab session #9 was the first one focused on the array concept, it dealt with creating and listing arrays. For the following lab session (#10) the level of difficulty was increased by dealing with swap and sequential search array operations.

Table 10. Categorical analysis for the student responses to the strategies to improve examples in Lab Session #9

Theme	Code	Definition	%	Representative Quote
Students struggled with specific elements within the examples	Nothing to Improve	The student thinks that the examples are fine the way they are presented	69	"*The examples given was perfect. I don't find any improvements needed." / "Nothing, the examples were good"*
	Less Comments	The student highlights the need to get rid of some of the comments since they have an impact on the code readability	6	*"Less Comments"*
	Complexity and Quantity	The student feels that it would be better to have more and more complex examples	15	*"Maybe harder ones" / "Not much, just detail and complex examples would help"*
Students suggested integrating more hands-on activities as part of the classroom approach	Better with Examples	The student thinks working with examples is a better approach than working from scratch	27	"*More examples*" / *"Nothing really, already enough material to help a novice like me"*
	In-class activities	The student thinks that the class activities should be focused on practical activities (design and programming activities)	6	*"More of class time is necessary to fully understand this language" / "Know how to build array"*
	Better without Examples	The student does not consider the examples as having helped her/him to solve the assignment	3	*"In order to remember how to write the code, I feel we should practice writing code (not typing), i.e., the methods, etc. until we know them."*

Table 11. Categorical analysis for the student responses to the strategies to improve the laboratory sessions in Lab Session #9

Theme	Code	Definition	%	Representative Quote
Students' suggestions to laboratory sessions	No suggestion	The student thinks that the laboratory sessions are fine the way they were carried out	67	*"No suggestions" / "I enjoy these labs immensely. I have no suggestions"*
	Logistic Improvements	The student feels that the laboratory session could be improved by having more time, different levels of difficulty, or more teaching assistants.	9	*"Maybe the instructor could walk us through the code that is already provided so that we have a better understanding of what we are going into. " / "Pretest and posttest during a lab adds stress to an inherently stressful situation"*
	Exploring examples	The student believes that exploring examples would help her/him to better understand the concepts before starting to build a program from scratch	12%	*"I like the way it was taught this week and last week" / "Perhaps more code demonstrations."*
	Better without examples	The student does not consider the examples as having helped her/him to solve the assignment.	12%	*"I prefer building programs from scratch, as I understand my own code better."/ "Writing code myself is the best way to improve my skill, at least for me".*

Table 12. Descriptive statistics of learning and perception scores grouped by commenting styles in lab session #9

Commenting Style	Pretest (%)		Posttest (%)		Lab Score (%)		Time to Complete (min)		Ability (%)	
	Mean	SD	Mean	Mean	Mean	SD	Mean	SD	Mean	SD
1. Detailed (N=6)	61.11	32.35	66.67	36.89	100	0	74	8.07	72.22	30.63
2. Basic (N=6)	45.83	42.41	54.16	36.04	93.33	7.53	80.33	10.60	53.70	25.98
3. Unclear (N=3)	33.33	28.87	61.11	41.94	65	56.34	72	8.66	85.19	16.97
4. Relevant (N=3)	86.11	24.06	77.78	38.49	98.33	2.89	89	12.49	85.19	13.98

4.3 Session #10: The student will create an algorithm to perform a sequential search and switch array elements

4.3.1 Quantitative Data

For this last session, the experimental and control groups were switched after lab session #9's configuration. Thus, the experimental group became the control group (N=16), while the control group became the experimental one (N=14). Table 13 shows descriptive statistics for the learning measures of these groups. Significant differences were found between pretest and posttest measures for the non-experienced students $t(12)=-2.14$, $p=0.053$ (one tailed t-test). With an average increment of 25%, students in the experimental group showed a significant change in the posttest learning measure as compared to the pretest. The result suggests that students in the experimental condition, with no previous programming experience, took advantage of the examples to increase their understanding about sequential search in arrays.

Regarding the perception measures, students in the experimental condition showed a positive perceived ability (Mean = 80.74%; SD = 16.38%) as compared to the neutral perceived ability presented by the control group (Mean = 65.93%; SD = 20.67%).

Table 13. Descriptive statistics of student learning scores in Lab Session #10

Test		Group		Programming Experience	
		Control (N=16)	Exper. (N=14)	Yes (N=8)	Yes (N=7)
Pretest (%)	Mean	56.64	66.67	75.78	56.25
	SD	25.36	15.25	11.29	12.50
Posttest (%)	Mean	64.06	77.50	73.21	81.25
	SD	25.87	22.76	16.81	27.55
Lab Score (%)	Mean	78.13	80	86.25	72.86
	SD	40.04	35.25	35.03	36.84
Time (min)	Mean	110.31	101.40	91.50	112.71
	SD	8.55	22.03	14.91	23.25

4.3.2 Qualitative Data

The open-ended questions asked at the end of the lab session #10 were analyzed following the codes and themes found on the previous lab sessions. On this iteration, fewer suggestions

concerning changes were made. Students highlighted that "These examples were clearer than in the past." Moreover, none of the students suggested that there was a need to reduce the comments or to change the quantity/complexity of the examples. Results are summarized in Table 14.

Regarding the lab sessions, the different perspectives about the preference of using/not using worked examples continued. Four students (13%) mentioned that they wanted to continue with examples, while two students (7%) preferred working from scratch. Three students (10%) talked about logistics, such as more time for the lab sessions or more lab sessions for specific topics. Seventeen students (57%) made no suggestions.

Finally, some students seemed worn out by the research process and complained about the time the pretest and posttest took from the session: "I don't have a problem with it but these in lab quizzes take away time from the overall lab and if they are a little sooner, then they might have trouble finishing lab in time." In fact, the complexity of this lab as well as the time taken to solve the tests made this lab session the longest in terms of the time to be completed. Therefore, only four students wrote the comments in the code. The distribution of these students for commenting styles was: (1) Detailed (two students); (2) Basic (one student); (3) Unclear –no comments- (ten students); (4) Relevant (one student).

4.3.3 Quan + Qual

Since the sample size became too small in this lab session, descriptive or inferential statistics were not calculated; however, to identify whether the trend that came from lab sessions #8 and #9 continued, the lab score for the three students in the detailed and relevant commenting styles were checked. All three students got a score of 100% , thereby confirming the trend.

5. DISCUSSION

This study explored the use of worked examples to support programming activities as part of an introductory course. Specifically, this study explored two questions and findings are discussed below.

5.1 How can worked examples be effectively designed to introduce programming concepts to novice learners?

Three laboratory sessions were used to introduce programming concepts using worked examples. The design and implementation of the worked examples were iteratively improved using students' suggestions and validated through learning assessments. The structure of the examples followed the principles suggested by Atkinson [5] that included: a problem statement, a procedure for solving the problem, and auxiliary representations of the problem and solution.

Two examples were used to scaffold the learning process in each session. The problem statement consisted of a single programming task aligned to the learning objective of the lab session and embedded within the problem set. The solution was represented in multiple forms including textual, graphical, and computational representations. All the representations were aligned with each other. A self-explanation task was also included as part of the assignment using written comments within the code to engage the students in the process.

The feedback from open-ended questions was useful for improving the examples. The main component of these edits was the elimination of complex explanations within the code that could generate additional cognitive load to students. In fact, the examples with the simplest comments (lab session #10) were the ones that showed significant differences. Some other changes were included such as: (1) avoiding the use of complex mathematical symbols; (2) increasing the complexity of the examples; and (3) aligning them to the problem assignments.

Only the last laboratory session (#10) presented significant differences in learning gains for students with non-programming experience. This result is aligned to what is suggested by Atkinson et al. [5] in that the worked examples approach may be useful for novices in an initial skill-acquisition stage such as analogy or abstract rules of learning [4].

Table 14. Categorical analysis for the student responses to the strategies to improve examples Lab Session #9

Theme	Code	Definition	%	Representative Quote
Students struggled with specific elements within the examples	Nothing to Improve	The student thinks that the examples are fine the way they are presented	61	*"They seem fine"/"Examples are fine"*
	More Detail	The student suggests increasing the level of detail in the examples or exercises.	6	*"More descriptions on how to reverse the array" / "Describe in more detail what the questions asking"*
Students suggested integrating more hands-on activities as part of the classroom approach	Better with Examples	The student thinks working with examples is a better approach than working from scratch	27	"*More examples*" / *"Nothing really, already enough material to help a novice like me"*
	In-class activities	The student thinks that the class activities should be focused on practical activities (design and programming activities)	6	*"More of class time is necessary to fully understand this language" / "Know how to build array"*
	Better without Examples	The student does not consider the examples as having helped her/him to solve the assignment	3	*"In order to remember how to write the code, I feel we should practice writing code (not typing), i.e., the methods, etc. until we know them."*

On the other hand, expert students with prior programming experience did not benefit from the examples, perhaps because they may have already developed a mental model [19]. For the rest of the sessions (#8 and #9), we speculate that the examples were unclear because they had too many comments included within the code. After students' suggestions, the examples were refined with simpler comments. Another possible explanation can be related to the time students need to get used to this new pedagogical approach. The worked examples approach was only introduced starting on lab session #8. Hence, the students were already used to a different problem solving approach. Moura [13] experienced this phenomenon and highlighted that students needed some time to get used to the tool she used for the worked examples. After that time, students performed better. Finally, the small sample size also made it difficult to find significant differences.

Regarding the perception constructs, novice students perceived their ability to solve various computing-related tasks to be significantly higher than those students with programming experience (in lab session #8). This, however, changed over time and a non-significant difference was found between experienced and non-experienced programmers for the rest of the iterations. The result suggests that, as the examples were improved, students with no previous experience were better able to take advantage of them. This is also suggested by the perceived ability of the students from the experimental group in the last session (80.74%), which was higher than the control group.

The worked examples approach generated a separation between those students who enjoyed exploring and learning from them and those who preferred to build the whole program themselves. From the students' responses from any of the sessions, of those who mentioned that they preferred coding from scratch, 75% identified themselves as experienced programmers. This is aligned with the rest of the findings and the literature suggesting that worked examples are more useful for novice learners than for expert ones [4, 5].

5.2 How do students self-explain worked examples when approaching a solution to a programming assignment?

Commenting on the code was used to encourage students' self-explaining process for the examples. These comments were grouped as four commenting styles: (1) Detailed; (2) Basic; (3) Unclear; and (4) Relevant (see Table 8 for a full description). Although non-significant differences were found between the groups, valuable insights were identified. First, as suggested by Chi et.al. (1989), students with a deeper self-explaining process (either (1) Detailed or (4) Relevant) performed better in all the lab sessions. Students with an incomplete self-explanation process appear to not fully understand the problem solving approach and, therefore, are unable to solve similar problems by analogy. Chi and collaborators [6] called this effect the self-explanation effect and enumerated four differences between students who were able to take better advantage of the examples than students who passively explored the examples. Trends identified in [6] were (1) high performers presented more self-explanations while studying examples; (2) "Poor" performers did not perform enough self-monitoring activities such as "I can see now how they did it"; (3) High performers referenced less to the examples when solving another problem than "poor" performers; (4) The "poor" performers self-explained more during the problem solving than the high performers who preferred to do it during the example exploration.

The second insight is that students who did not include any comments reported a higher perceived positive ability than those students who wrote very simple comments. We speculate that these students felt confident about their abilities and, therefore, did not want to spend time understanding another approach; however, they did not perform as well as students who wrote thorough comments.

The main limitation of the study is the small sample size constrained by the course size. Therefore, the significance of the differences found in this study lies in the qualitative data regarding students' recommendations, perceptions, and commenting styles. Another limitation is that the worked examples approach began in lab session #8. This means that the students had been exposed to seven previous sessions with a different approach. This may have generated a negative reaction in some students who preferred to work in a more familiar way.

6. IMPLICATIONS

6.1 Implications for Teaching

The use of Atkinson's instructional principles to design worked examples has been identified as useful in situations where novice learners seem to take more advantage of this technique. Expert learners may have already acquired mental models in the thematic area that provided them with the necessary tools for problem solving.

The identification of intra-example, inter-example, and interacting-with-the-learning-environment features of worked examples can provide a framework for instructors to effectively design their worked examples. Specifically, the intra-example features used in this study presented several requests by the students to keep simple explanations, especially when they are integrated into the code. Students often mentioned that many comments within the example code decreased readability. Thus, the use of at least two different examples with a good alignment with the assignments is the main inter-example feature that should be considered.

Finally, for programming activities, requiring students to write comments within the code can be useful as a self-explanation process; however, to take full advantage of this process, it is important to encourage students to write detailed comments or to highlight relevant conditions by describing boundaries and the consequences of their solutions.

6.2 Implications for Learning

Results from this study suggest that students who described relevant conditions along the code, as well as details in the way the code worked, performed better than those students who commented on the code superficially or did not self-explained it at all. Several studies have demonstrated that a passive approach to studying worked examples has no impact on learning as compared to problem-solving instruction (Chi et. al., 1989; Atkinson et al., 2000). The reason for this could be a lack of understanding resulting from not actively engaging with the examples.

Chi and colleagues [6] suggested that the examples are not always completely clear, so the students have to engage in a self-explanation process allowing them to identify the relevant aspects of the solution. Thus, a self-explanation should contain four aspects that depict an understanding: (1) the conditions of

application of the actions; (2) the consequences of actions; (3) the relationship of actions to goals; and (4) the relationship of goals and actions to natural laws and other principles. In this study, the "Detailed" and "Relevant" commenting styles contained all these characteristics while "Basic" or "Unclear" commenting styles contained only one of these features, (e.g. the conditions of application of the actions) if any of them at all. Furthermore, a good understanding of the example can lead to more proficient problem-solving skills, while poor understanding may lead to a continuous reference to the example while trying to solve another problem.

7. CONCLUSION

The use of worked examples to scaffold programming and algorithm design learning has been evaluated. Different instructional design elements were assessed in order to identify effective design characteristics for worked examples. Multiple representations of the solution, including textual, graphical and computational representations, were employed. Writing in-code explanations as simple sentences enhanced code readability and improved students' perceptions about the examples. Moreover, encouraging students' self-explanation process by asking them to comment within the code helped the students to actively engage with the examples. Specific suggestions include encouraging students to write detailed comments as opposed to superficial ones in order to take advantage of the examples. This approach seems to be useful for novice students who did not have previous experience in programming.

The contribution of the study is the detailed description of the implementation of worked examples in a programming context. It includes the use of multiple representations as well as the use of comments within the code as a self-explanation process.

8. LIMITATIONS AND FUTURE WORK

The main limitation of this study is that the learning outcomes for each iteration were different. Thus, the changes implemented based on the results were not evaluated in exactly the same context. Therefore, future work will explore the effect of these recommendations for these three lab sessions.

Next steps also include the design of additional examples using instructional principles of worked examples [5] as well as students' suggestions in this process. Future instruction should also encourage students to carry out a thorough self-explaining process that may lead them to an understanding of the examples. This can be accomplished either through incentives or by means of extended training.

9. ACKNOWLEDGMENTS

This research was supported in part by the U.S. National Science Foundation under the award #EEC1329262.

Authors would also like to thank Guity Ravai for her assistance in reviewing the assessment instruments and facilitating the implementation of the learning materials.

10. REFERENCES

[1] [NRC]. 1999. Being fluent with Information Technology: National Academy Press.

[2] [PITAC]. 2005. "Computational science: ensuring America's competitiveness," President's Information Technology Advisory Committee (PITAC), vol. 27

[3] [WTEC]. 2009. "International assessment of research and development in simulation-based engineering and science" World Technology Evaluation Center, Inc., Baltimore, Maryland.

[4] Anderson, J. R., Fincham, J. M., and Douglass, S. 1997. The role of examples and rules in the acquisition of a cognitive skill. Journal of Experimental Psychology: Learning, Memory, and Cognition, 23, 932-945.

[5] Atkinson, R.K. Derry, S. J., Renkl, A., and Wortham, D. 2000. Learning from Examples: Instructional Principles from the Worked Examples Research/ Review of Educational Research. Summer 2000, Vol. 70, No. 2, pp. 181-214.

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

[7] Cuny, J., Snyder, L., and Wing, J. M. 2010. Demystifying Computational Thinking for Non-Computer Scientists. Work in progress.

[8] College Board. 2011. CS Principles. http://www.collegeboard.com/prod_downloads/computerscience/Learning_CSPrinciples.pdf

[9] du Boulay, B. 1989. Some difficulties of learning to program. In E. Soloway & J.C. Spohrer (Eds.), (pp. 283–299). Hillsdale, NJ: Lawrence Erlbaum.

[10] Gray, S., St. Clair, C., James, R., and Mead, J. 2007. Suggestions for graduated exposure to programming concepts using fading worked examples. Proceedings of the third international workshop on Computing education research - ICER '07, 99.

[11] Guzdial, M. and Robertson, J. 2010. Too much programming too soon? . Communications of the ACM, Volume 53 Issue 3, March, 2010.

[12] Kirschner, P. A., Sweller, J., and Clark, R.E. 2006. Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching, Educational Psychologist, Vol. 41, Iss. 2.

[13] Moura, I. C. 2013. Visualizing the Execution of Programming Worked-out Examples with Portugol. Proceedings of the World Congress on Engineering 2013 Vol I, WCE 2013, July 3 - 5, 2013, London, U.K.

[14] Paas, F., Renkl, A., and Sweller, J. 2004. Cognitive load theory: Instructional implications of the interaction between information structures and cognitive architecture. Instructional science 32, 1-8.

[15] Paas, F., Renkl, A., and Sweller, J. 2003. Cognitive Load Theory and Instructional Design: Recent Developments. Educational Psychologist, 38(1), 1–4.

[16] Pirolli, P. and Recker, M. 1994. Learning strategies and transfer in the domain of programming. Cognition and Instruction, 12, 235–275.

[17] Renkl, A., Atkinson, R. K., and Große, C. S. 2004. How Fading Worked Solution Steps Works – A Cognitive Load Perspective. Instructional Science, 59–82.

[18] Rist, R.S. 1995. Program structure and design. Cognitive Science, 19, 507–562.

[19] Robins, A., Rountree, J., and Rountree, N. 2003. Learning and Teaching Programming: A Review and Discussion, Computer Science Education, Vol. 13, No. 2, pp. 137–172.

[20] Rogalski, J. and Samurc¸ay, R. 1990. Acquisition of programming knowledge and skills. In J.M. Hoc, T.R.G. Green, R. Samurc¸ay, & D.J. Gillmore (Eds.), Psychology of programming (pp. 157–174). London: Academic Press.

[21] Barab, S. and Squire, K. 2004. Design-based research: Putting a stake in the ground. Journal of the Learning Sciences, 13(1), 1–14.

[22] Sweller, J. 1988. Cognitive load during problem solving: effects on learning. Cognitive Science 12, 257-285.

[23] Teddlie, C., and Tashakkori, A. 2006. A general typology of research designs featuring mixed methods. Research in the Schools, 13(1), 12-28.

[24] Wing, J. 2006. Computational Thinking. Communications of the ACM, 49, 3, 33-35