USING A HYBRID APPROACH TO FACILITATE LEARNING INTRODUCTORY PROGRAMMING

Assist. Prof. Dr. Ünal ÇAKIROĞLU
Karadeniz Technical University, Computer Education & Instructional Tech. Dept.,
cakiroglu@ktu.edu.tr

ABSTRACT
In order to facilitate students' understanding in introductory programming courses, different types of teaching approaches were conducted. In this study, a hybrid approach including comment first coding (CFC), analogy and template approaches were used. The goal was to investigate the effect of such a hybrid approach on students' understanding in introductory programming A quasi-experimental design and one control group (CG, N =38) and one experimental group (EG, N = 38) were used. While the control group was taught in the traditional way, the experimental group received another instructional package which included the hybrid approach. Three open ended questions were administered as a pretest and a Programming Knowledge Test (ProKT) was administered as a posttest. The Posttest results were examined in two domains (conceptual understanding and problem solving). In addition, the observations made in the EG classroom were interpreted as qualitative data. While there was no statistically significant difference between two groups in the pretest scores, EG students performed better than the CG students in problem solving domain of posttest. Observations and posttest results showed that the EG students were better in remediating the deficiencies especially in problem solving, in addition to basic programming concepts and language features. Based on these results; it was concluded that integrating three different approaches together has positive effects on facilitating students' development of introductory programming knowledge.
Keywords: Analogy, Templates, Comment First Coding, Teaching Programming Languages

INTRODUCTION

Wiedenbeck & Ramalingam (1999) defined programming as a process which includes a variety of cognitive activities, and mental representations related to program design, program understanding, modifying and debugging. In addition, Bayman & Mayer (1988) specified that programming involves syntactic, conceptual and strategic knowledge. The syntactic knowledge includes programming languages' specific facts and rules, conceptual knowledge concerns programming structures and principles, and strategic knowledge is related to applying general problem solving skills. The programming process is a complex process which has two main phases (problem solving and implementation) summarized in Figure 1.

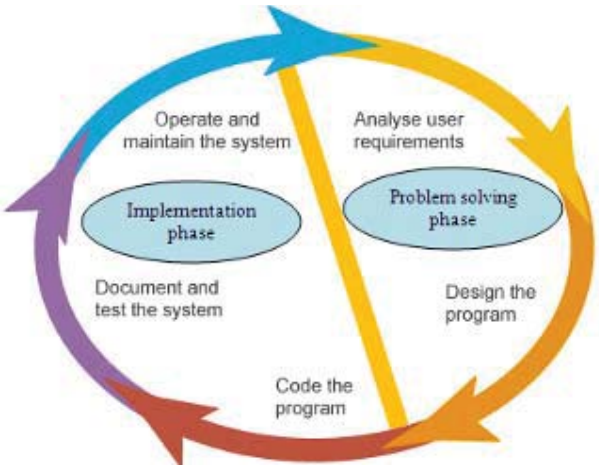


Figure 1: Programming Process

The steps and the processes of programming in the implementation and problem solving phases are specified in Table1.

Table 1: Phases of programming process

	Programming Phase	Processes done in this Phase
Phase1	Analyze user requirements	Problem is defined; what the inputs and the outputs should be, and the operational parameters within which the system is expected to work.
	Design the Program	Solution to the problem is designed by defining a logical sequence of steps that will achieve each of the stated system objectives such as algorithms or flowcharts.
	Implementation Phase	**Processes done in this Phase**
Phase2	Code the Program	The algorithms are translated into a programming language with concrete solutions.
	Document and Test	Providing technical references to inform the user about the features of the software and how to use it.
	Operate and maintain the system	Monitor the performance of the system over time to ensure that it is behaving as expected.

Introductory programming courses have become mandatory in computer science and informatics programs. In Turkey, also, in computer and instructional technologies departments, introductory programming courses are mandatory in the second year both in the first and second semesters. The objectives of the courses are generally to understand the first three steps (analyze, design and code) of the programming process. These steps are indubitably complex for novices to understand and this complexity causes some deficits in learning programming. Ismail, Ngah & Umar (2010) defined the reasons for these deficits as being a lack of problem solving skills, lack of analytical thinking skills, and a lack of programming conceptual understanding.

RELATED STUDIES

Novices may be investigated in three categories. The first category includes really poor students who do not understand the basic concepts. The second may understand basic concepts if the teachers use effective teaching approaches. The third are those who can easily grasp the nature of programming concepts (Dunican, 2002). Introductory programming courses generally aim to enhance students in the second category, allowing them to progress to the third. Although different types of teaching approaches were used to promote novices' performances, poor rates of student success have been indicated in many studies (Barg et al, 2000; Carbone & Sheard, 2002; Hagan & Macdonald, 2000; Stein, 1999; Williams & Kessler, 2000). In this context; a number of challenges have been identified (Dunican, 2002; Jenkins, 2002; McCracken et al, 2001; Proulx, 2000). Winslow (1996) pointed out that some students can solve the problem manually but they have trouble translating solutions into equivalent computer programs. It is known as transferring step-by-step problem-solving from a natural language into a program. In addition, Weigend (2006) observed that students may find a mental or practical solution to a problem but they fail to write a correct program for solving the problem. Another deficit is a lack of general problem solving skills because the students are the product of an educational system that does not have this kind of problem solving module included in any of their subjects. Another problem concerns concretizing components such as variables, data types, memory etc. Mannila (2007) stated that there seems to be a general lack of attention to program comprehension skills in education. If students cannot understand the code presented, they may shape their own conceptions and strategies and that is certainly something instructors want to avoid. In addition; De Raadt (2007) reviewed Australasian research studies concerning novice programmers. These studies have shown that novices are not performing at expected levels and many novices have only a basic knowledge of programming.

In this context; Ismail, et al (2010) reviewed several studies and made note of the most difficult issues for students, which are summarized in Table 2.

Table 2: Difficulties in Programming knowledge

Difficulties	Programming knowledge
Deficiencies in general programming constructs	Basic programming concepts
Using ineffective coding and designing techniques	Language features
Unable to analyze problems and using ineffective problem solving techniques	Problem solving

Moreover, a multinational study among university students have shown that there is a worldwide difficulty in mastering computer programming (McCracken et al, 2001).

Some different approaches have been conducted for eliminating the difficulties. Miliszewska & Tan (2007) discussed some of the difficulties experienced by first year programming students. They also reported on the first stage of a project designed to develop a balanced approach for teaching. Hui Hui & Umar (2011) have investigated the effect of metaphors and pairing activity in programming process. They found that metaphors have assisted learners in developing better conceptual understanding by enabling them to link known to newly acquired abstracts and that pair programming cultivates peer discussions. Another study suggested a teaching model integrating cognitive conflict and program visualization. They found the teaching model is potentially effective in enhancing engagement with learning materials and may therefore help novice programmers to understand basic concepts better (Ma, Ferguson & Wood, 2011).

Although some new approaches have been tested; the difficulties in learning programming show that there is an urgent need to find new ways for better student learning outcomes (Thuné & Eckerdal, 2009). This is not easy to achieve unless the new teaching approaches are implemented and adequate pedagogical techniques are utilized.

Purpose of the study

The difficulties in teaching programming show that there is still a need for finding new ways of attaining better student learning outcomes. This paper focuses on a hybrid teaching approach by combining three teaching approaches taken to teach introductory programming to novices. The main question discussed in the study is: Does using hybrid approach have an effect on conceptual understanding and the problem solving domains of introductory programming? Also, the findings were interpreted to estimate how the approaches affected the two domains of programming.

METHODOLOGY

Selecting Programming Paradigms for Novices

Since problems can be solved in different ways, choosing the best paradigm for problem solving is not easy. Besides, there is still not a consensus on a programming paradigm for introductory courses. By evaluating the advantages and disadvantages of paradigms, Vujosevic-Janici & Tosi'c (2008) pointed out that the most suitable paradigms for introductory students are procedural, procedural part of the object-oriented, and the event-driven programming paradigms. In this context, introductory courses should focus on general programming ideas and concepts, while considering both basic and more advanced concepts. Siegle (2009) indicates that imperative and event driven programming paradigms appear to be the dominant paradigm in most introductory courses.

In this study, an imperative paradigm underlying the basic problem solving techniques was introduced first. After the imperative paradigm; Delphi, including the event driven paradigm, was taught. By using the event driven paradigm, students used events of the programming language environment provided to make high level abstractions. So they were only responsible for writing codes to solve the problem, and did not have to struggle with other details.

Teaching Approaches in This Study

In order to facilitate the development of programming knowledge, different approaches (CFC, analogies and templates) were introduced at different phases of the programming process. Figure 2 illustrates the phases and the approaches selected and used in treatment.

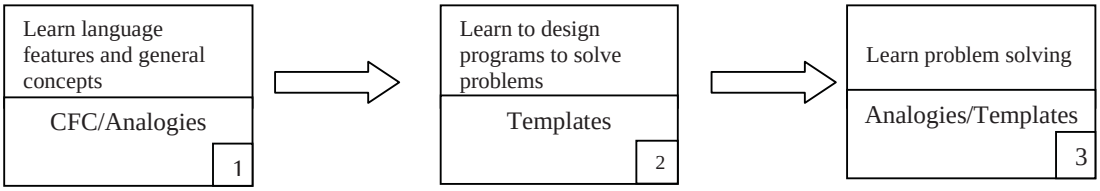


Figure 2: The approaches used in treatment

CFC : Although learning the syntax or semantics of a statement individually is not so hard, combining them is usually difficult for novices. Also, solving problems on paper or finding the solution mentally is often easy, but generally students have trouble translating their thoughts into an appropriate programming code (Sengupta, 2009). One approach for facilitating the overcoming of this problem is to use comments before coding programs known as CFC. Generally comments are used for making the source code easier to understand. It means coding that is like talking. Hence, in order to transfer a solution of a problem step by step, directly from a natural language into a program, the CFC method is used in this study.

Example: *"Decomposing words from a sentence (eg. "I went to the cinema")."*

1//Input the sentence.
2//Put "@" character to the end of sentence.
3//Locate the first space character.
4//Set the index of space character as end of the word.
5//Set the index of space +1 as the beginning of next word.
6//Repeat 3-6 until the last character "@".

Analogy : In the programming process analogy is performed to construct new concepts by using the existing ones that the students are already familiar with. As it is known, analogy plays a significant role in problem solving, decision making, memory and creativity (Gentner, 2000). In programming, sometimes problems recall other problems which students have seen or solved before. If students remember the previous solution method, they may use it also to solve the current problem. This may be defined as drawing an analogy between the solution methods. Analogy can also be used to illustrate statements or general programming concepts. For example, Dunican (2002) describes several analogies: the use of children's toys to teach assignment statements, the use of boxes to determine the smallest and largest number in a list, and the use of a leaflet distributor to explain the concept of array manipulation.

Templates : Linn & Dalbey (1989) suggested using templates to facilitate implementation and problem solving. According to Rist (1991) programming is a process of implementing basic plans which form more complex plans when combined. To achieve the goal, a programmer can search for templates. Templates perform useful functions (eg., sorting a list of numbers or addition of a set of numbers) which can be considered as units of programming knowledge which are designed by experts who have had many years of experience addressing many different problems. Expert programmers know a great deal more than just the syntax and semantics of language constructs (Byckling & Sajaniemi, 2006; Rist, 1991). Templates may enhance novices' programming more effectively, in problem solving and also in the design phases. Interfaces, forms, lists, outputs or other components may be designed by using the experts' examples. In this sense, teachers generally indicate that getting support from experts' templates will be useful for students.

Example: *"Writing a program to find the frequencies of the words in a list."*

A student may have a plan in three stages used by an expert.
(1) read the words using the loop template,
(2) find the frequencies using the frequency template,
(3) use print template to print sorted word list.

These three approaches may be used either one by one or together in the program coding process. An example is shown in Figure 3 including the analogy and template approaches together. The example is a code for "sorting the given numbers in ascending order".

```
Procedure sort ()
var
List1: TstringList;
a,i,j:integer;
begin
 List1: Tstings.create;
 List1.items.add (Edit1.text);
 Analogy is used here (The tallest)
 Assign first student as tallest
 If the next one is taller than the first, swap the order (1, 2)
 Repeat this to the end of the list.
for i:= 0 to List1.count-1 do,
 for j:=i+1 to List1.count-1 do;
  If List1[i]>List1[j] then
begin
 Template is used here (Swapping)
  a:=List1[i]; List1[i]:=List1[j]; List1[j]:=a;
end;
```

Figure 3: A piece of code which Analogy and Templates used together

Research Design and Sample

The study was conducted over one semester in introductory programming courses at a computer and instructional technologies department to second year students as a quasi-experimental design. The participants enrolled in the course, one experimental group (EG) (with 38 students, 22 male, 16 female) and a control group (CG) (with 38 students, 20 male, 18 female) were assigned as naturally occurring groups in the study. The

distribution of the students' graduate schools was both similar. The researcher was also the tutor of both the groups.

Process

The intervention was conducted during course time through 5 lessons a week; 3 hours in the classroom and 2 hours in the lab, for 10 weeks, the details of which are shown in Table 3. Both of the groups were informed about language features, program design and problem solving in the courses. The courses included programming language constructs, algorithm development, interface design, general programming concepts (memory, data types, functions, procedures, and the basic data structures, I/O operations) and problem solving activities. The CG received traditional instruction which included teacher explanations, demonstrations, and simple problem solving examples. The CG students were frequently taught using 'chalk and board' lessons. They used text books and teachers' notes as references. In the lab they studied the examples and did their homework. EG students received the same subjects with another instructional package, which included a combination of three different approaches in teaching programming. These approaches were CFC, analogy, and the template, which were used in the different phases of the programming process in order to support student's learning. This kind of educational support is generally called 'scaffolding', Vygostky's term which suggests helping students by bringing them up to a higher level of understanding. The schedule for two groups is shown in Table 3.

Table 3: The schedule for two groups

Week	Subject	Teaching Methods	
		Control Group	**Experimental Group**
1	First program, compilation, syntax errors	Presented on PC screen	Presented on PC screen
2	Variables, data types, memory, and arithmetic expressions, conditional statements, complex conditions	Used direct teaching, presentation method in teaching arithmetic expressions and conditional statements	Used analogies in teaching variables, data types, memory and conditional statements
3	Loops (for, while, repeat)	Generally used direct teaching, gave examples from textbook.	Used different analogies in teaching for, while and repeat. Taught how to use CFC.
4	Arrays, character arrays and string processing, functions, procedures, parameter passing, problem solving	Used direct teaching, used presentations on showing array examples	Made analogies for expressing arrays, parameter passing and used several templates about arrays, functions and procedures. Also stimulated students to use CFC.
5	Recursion, problem solving	Wrote the recursion codes on the board and examined the codes line by line.	Used templates about searching, sorting and trees also used CFC codes in long examples. Used analogies when solving the problems.
6	Introducing the environment, forms and menu items	Presented the Delphi environment by projector	Presented the Delphi environment by projector
7	Input/Output on Canvas	Taught to design basic input output forms, message boxes	Showed simple message boxes then showed how to used experts' template forms and boxes.
8	Event handlers and methods	Showed on Delphi how to use Delphi events and methods, gave examples about the events of buttons, edit boxes.	Used analogies when teaching different methods
9	User interface design, reports	Showed simple interfaces based on forms, menus and boxes, showed only one type of reporting.	Used templates in interface design
10	Applications using visual components library, debugging and testing of Delphi programs	Students constructed examples by using different visual components in lab.	Students constructed examples by using different visual components in lab.

Instruments

The study began with the administration of three open ended questions to both groups of students as a pretest (see Appendix1). The pretest included questions about basic programming language features, basic concepts in programming and a simple problem. The pretest showed that the students had similar distributions in programming experience in both groups. The participants in EG were 16 inexperienced, 15 basic level, 5 intermediate level, and 2 advanced level of experience. The distribution in CG were 18 inexperienced, 14 basic level, 5 intermediate level, and 1 advanced level. The statistical analysis of the pretest also showed that there was no significant difference in students' programming knowledge at the beginning.

During the intervention, the researcher observed students in lessons for each group, with each group following the same subjects. In the last week of the intervention, the "Programming Knowledge Test" (ProKT) (see Appendix2) was administered to both groups as a posttest. The ProKT was designed through a programming knowledge evaluation model (Bayman & Mayer, 1988). According to their approach, programming knowledge may be examined in three main categories shown in Table 4. In this sense, from a technical point of view Govender & Grayson (2006) also specified three main aspects (data, instructions and syntax) for learning introductory programming.

Table 4: Categories of programming knowledge

Syntactic knowledge	Using statements syntactically correct
Programming Structure Knowledge	Give meaning to actions when programming executing, design solutions for programming problems.
Strategic Knowledge	Combining design, code and test knowledge with problem solving skills

Considering this model, the open ended questions related to programming knowledge were prepared by creating an evaluation test (McGill & Volet, 1997). The test was adapted to the Delphi considering the instructional objectives and common student conceptions reported in the evaluation test. Two raters first assigned the points for the questions individually, and then they discussed the questions with each other until they come to exact agreement on each item on the scale. It can be concluded that they shared a common understanding of the criteria on the scale.

Data Analysis

In the analysis of ProKT, the total score of each student in both groups as well as the mean score of each group were computed. This computation was performed after all test papers were examined and rated by two independent expert tutors as per the criteria (See Appendix 3); one has a 10 year programming past, the other has 12 years of experience as a programming instructor. As it is known, raters are often used when students' performances cannot be scored objectively as right or wrong but require a rating of degree (Stemler, 2004). The Pearson correlation coefficient between the raters' scores was r= 0.84 which is between (0.75-1) and can be considered as a strong positive correlation. The explanations sections for all questions in the test were also examined to determine why the students used the approaches to answer the questions.

The statistically significant difference between CG and EG was determined by using the independent t-test. Because there was no statistically significant difference between the pretest results, the posttest results were also compared using the independent t-test to address the effects of intervention. In addition, the observations were interpreted with quantitative data.

RESULTS

The mean scores of the pretest were found to be similar between the two groups: EG (M: 48.29; SD: 13.47) and CG (M: 50.52; SD: 11.96). Also, as shown in Table 5, there was no significant difference between the mean scores of the groups EG and CG (t(74): -7.66, p:0.446>0.05) according to the independent t-test results. This indicates the similar backgrounds of the students in both groups before the intervention.

Table 5: The results of t-test on pretest scores (EG and CG)

Groups	N	M	SD	df	t	P
EG	38	48.29	13.47	74	-7.66	.446
CG	38	50.52	11.96			

The rest of the results section organized as; comparison on posttest results, comparison in conceptual understanding and problem solving, observations during the procedure in EG, experts' reviews on students' posttests and clinical interviews on posttests.

Comparisons on posttest scores

The descriptive statistics for the data obtained from the posttest is presented in Table 6. An independent sample t-test was conducted with the data from all sections of the posttest. (Conceptual understanding (Q1 and Q2), Problem solving (Q3 and Q4). A statistically significant difference between the mean posttest (ProKT) scores of the EG (M: 58.90; SD: 15.26) and CG (M: 46.63; SD: 14.28) groups was found (t(74): 3.825, p: 0.000<0.05). This reflects that the EG students performed better after treatment considering all sections of ProKT.

Table 6: The results of t-test on posttest scores (EG and CG)

Groups	N	M	SD	df	t	P
EG	38	59.60	15.26	74	3.825	.000
CG	38	46.63	14.28			

Since the questions are related to the different domains of programming knowledge, the analysis was carried out by considering the questions one by one, and considering the two different domains. The independent t-test results for the two groups about the first three questions (Q1, Q2 and Q3) are shown in Table 7. Since Q3 and Q4 are related to the "problem solving" domain, they were analyzed individually and the results of Q3 are included in Table 7. As the computation showed that the distributions of variables do not show normal distribution, the Mann Whitney U Test was conducted within Q4 in order to determine the significance of the means of the scores of each group statistically. The results concerning Q4 are presented in Table 8.

Table 7: The results of t test on posttest scores (EG and CG) for (Q1, Q2, Q3)

Phases	Questions	Groups	N	M	SD	df	t	P
Conceptual Understanding	Q1	EG	38	14.03	2.28	74	0.413	.681
		CG	38	13.81	2.15			
	Q2	EG	38	17.76	4.89	74	1.631	.107
		CG	38	15.66	6.27			
Problem Solving	Q3	EG	38	16.05	5.47	74	3.594	.001
		CG	38	11.18	6.30			

In Q1 within "conceptual understanding" the descriptive results are; EG (M: 14.03; SD: 2.28) and CG (M: 13.81; SD: 2.15) in favor of EG and statistically (t(74)=0.681, p: 0.681 >0.05). The results for Q2 are: EG (M: 17.76; SD: 4.89) and CG (M: 15.66; SD: 6.27) in favor of EG; where (t(74)=1.631, p: 0.107>0.05). The result reflects that, when the scores about Q1 and Q2 for both groups is analyzed individually, no significant difference exists between EG and CG.

In Q3 within "problem solving", the t test results are: EG (M: 16.05; SD: 5.47) and CG (M: 11.18; SD: 6.30) in favor of EG, and statistically (t(74)=3.594, p: 0.001<0.05). According to the averages of scores for Q4, the results from the Mann Whitney U test shows that there is a prominent difference in favor of group EG. EG (M: 11.76), CG (M: 5.97).

Table 8: Mann Whitney U test result for Q4

Phases	Questions	Group	N	Mean Rank	Sum of Ranks	U	p
Problem Solving	Q4	EG	38	48.46	1841.50	343.500	.000
		CG	38	28.54	1084.50		

A significant difference among groups was found in favor of EG; U=343.5, (p: 0.000<0.005). So, after examining the scores of Q3 and Q4, it was found that, statistically, the mean scores taken from each question respectively illustrate that both have significant difference favoring EG.

Comparison in "conceptual understanding" and "problem solving"

Q1 and Q2 were considered together to determine "conceptual understanding" in programming knowledge. Q3 and Q4 were also analyzed together in order to determine whether or not there was significant difference between the groups in the "problem solving" domain. The independent t-test results are presented in Table 9.

Table 9: The results of t test on posttest scores (EG and CG) for (Q1- Q2 and Q3-Q4)

Phases	Question	Groups	N	M	SD	df	t	P
Conceptual Understanding	Q1-Q2	EG	38	31.78	5.88	74	1.507	0.136
		CG	38	29.47	7.42			
Problem Solving	Q3-Q4	EG	38	27.82	11.26	74	4.546	0.000
		CG	38	17.16	9.05			

The mean scores regarding Q1 and Q2 together were EG (M: 31.78; SD: 5.88) and CG (M: 29.47; SD: 7.42) in favor of EG. Posttest results for (Q1 and Q2), did not show significant difference in the mean scores between groups for "conceptual understanding"; the statistical result was t(74)=1.507 and p>0.05). The descriptive results were in favor of EG, that is EG (M: 27.82; SD: 11.26) and CG (M: 17.16; SD: 9.05). Statistically, a significant difference exists in the "problem solving" domain of programming knowledge between groups in favor of the EG, which are t(74)=4.546, (p:0.000 <0.05). As a result, students in the EG attained higher levels of programming knowledge than did the CG students. It is important to notice that the EG students' scores are significantly greater than the CG ones in the (Q3 and Q4) "problem solving" domain. This may be interpreted as an indication of the effectiveness of the use of the hybrid approach. In contrast with the "problem solving" domain; there was no significant difference between EG and CG in the "conceptual understanding" domain. The average scores of the EG and CG from posttest sections are shown in Figure 4.

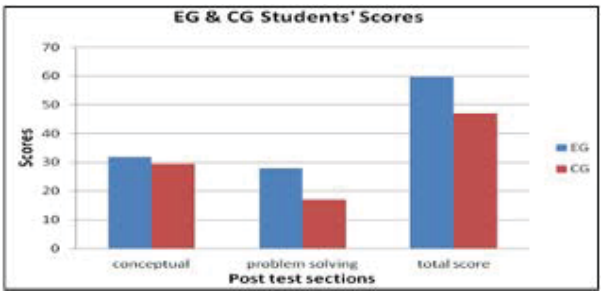


Figure 4: EG and CG students posttest averages scores

Observations during the procedure in EG

During the procedure the researcher observed and noted the student behaviors in the EG. The main questions were, "Can students use the new approaches?", "Do they understand where they can use the approaches?" and "Is there any difference between EG and CG groups learning programming process?" The sequence of used approaches was planned through the curriculum in the sequence of language features, basic programming concepts and problem solving, as shown in Table2.

During the intervention the researcher generally observed that students easily grasped the use of three new approaches. At the beginning they used CFC in informal ways within their own sentences, but they learnt to develop CFC sentences in time. After six weeks, the instructor designed problem solving activities. In addition, during the process, some analogies in programming concepts (memory, arrays, variables, stacks, etc.) were taught to the EG. Analogies took some more time for instructor to teach concepts by using them. Analogies were helpful to EG students but sometimes they still made mistakes in using abstract concepts like memory, disk allocations etc. Some of the students could also use suitable templates for their programs. A few of them tried to memorize some templates, but instructor made them to use templates by referring from experts codes. In problem solving activities, students sometimes failed to engage in planning the problem solving process. The main problem was decomposing problems into sub problems that they learnt to use templates for solving these problems.

Experts' reviews on students' posttests

Student posttest responses were evaluated by two experts as per the criteria in Appendix3. The approaches used in the answers are noted on Table 10 and Table 11.

Table 10: The distribution of approaches used by EG students in posttest answers (1-18)

Student (The first 18 scores)	S37*	S8*	S7*	S5*	S19*	S6	S18*	S14*	S29	S16	S36	S30*	S4	S34*	S28	S9	S20	S21
CFC	Q2 Q3 Q4	Q2 Q4	Q3 Q4	Q3 Q4	Q3 Q4	Q3 Q4	Q2 Q3 Q4	Q2	Q3 Q4	-	Q2 Q3	Q3 Q4	Q3 Q4	Q3 Q4	-	Q3 Q4	Q3 Q4	Q2 Q3
Analogy	Q2 Q	Q2 Q	Q2 Q	Q2 Q	Q1 Q	Q4	Q3 Q	Q2 Q	-	Q2 Q	Q2 Q	Q2 Q	Q3	Q2 Q	-	-	- Q3	-

Template	3 Q4 Q3	4 Q3	3 Q3 Q4	3 Q4 Q3 Q4	3 Q4 Q2 Q3	Q4	4 Q3 Q4	3 Q4 Q3 Q4	Q4	3 Q4 Q4	4 -	3 Q4 Q3 Q4	Q3	3 Q4 Q3	-	-	Q4	-
Post test score	90	88	85	78	77	75	75	70	70	68	68	67	65	65	62	60	60	60

*: Student used three approaches at least 5 times.

Table 11: The distribution of approaches used by EG students in posttest answers **(**19-38)

Student (The last 18 scores)	S31	S3	S1	S2	S23*	S24	S26	S32	S22	S25	S38	S12	S13	S15	S33	S10	S11	S17	S35	S27
CFC	Q3 Q4	Q2 Q3 Q4	-	Q4	Q3 Q4	-	Q3 Q4	-	Q3 Q4	-	Q3 Q4	Q3 Q4	-	Q4	Q2	Q3	Q4	-	Q3 Q4	Q2
Analogy	Q3 Q4	-	-	Q3 Q4	Q2 Q3 Q4	Q2	Q4	-	-	-	Q3	Q3 Q4	-	-	Q2 Q4	Q2	Q1 Q2 Q4	Q2	-	Q3
Template	-	Q4	Q3	Q4	Q4	-	Q2	Q4	-	-	Q2 Q4	-	-	Q3	-	-	-	Q3 Q4	-	-
Post test score	60	58	55	55	55	55	55	53	50	50	48	45	45	45	45	43	40	35	35	30

*: Student used three approaches at least 5 times.

It can be seen from Table 10 and Table 11 that there were 19 students who used all of the three approaches and only S13, S25 and S28 did not use any of the approaches. Table 10 and Table 11 show that S37, S8, S7, S5, S19, S18 and S14 (Table 10) and S23 (Table11) used all of the three approaches at least 5 times. As can be seen, their total scores are at least 55. In addition, the first 8 students all used these three approaches except S6 who used only Q3 and Q4. These statistics present some evidence that students who used all of the approaches performed better than others. Another statistical data was obtained by determining the approaches' frequency of use related to the questions. This may also give an idea of the relationship between students' total score and the use of approaches in the posttest. So Figure 5 shows the frequencies of approaches in descending order of scores.

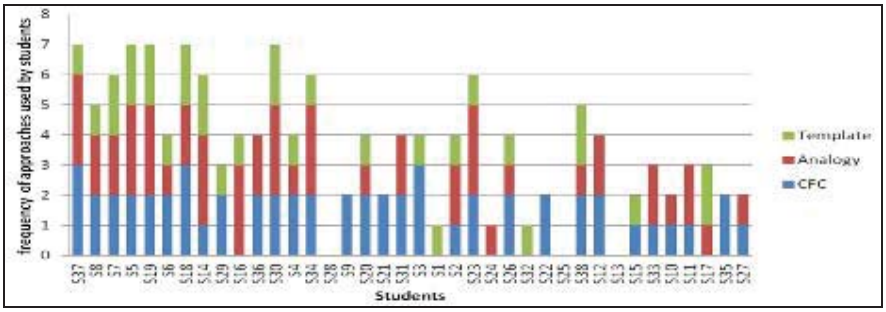


Figure 5: The frequencies of use of approaches (Listed from highest score student to lowest)

The frequencies of approaches used in students' answers are illustrated on Table 12.

Table 12: Frequencies of approaches used student answers in EG

	Questions				
Used Approach (f)	Q1	Q2	Q3	Q4	Total
CFC	-	9	24	26	60
Analogy	2	18	19	21	58
Template	-	4	17	21	42
Average Score	14.02/16	17.76/24	16.05/24	11.76/36	

As seen from Table 12, in Q1, only 2 students used analogy. In Q2, 9 students used CFC, 18 students used analogy and 4 of them used the template approach. In Q3; 24 students and, in Q4; 26 students used CFC. 19 students provided solutions using analogy in Q3 and this approach was preferred by 21 students in Q4. Templates were used 17 times in Q3 and 21 times in Q4. The total frequencies of use of approaches may give some idea of the order of students' preferences to the approaches, which are, in descending order, CFC, analogy and template.

Clinical Interviews on Posttests

In order to obtain more data regarding the effects of the approaches; clinical interviews (CI) were conducted with 5 students. These students were (S14: used both of three approaches, S31: used CFC and analogy, S9: used only CFC, S3: used CFC and templates, The passages from interviews related to their answers are shown in following tables (Table 13, Table 14, Table 15, Table 16)

Table 13: Examples of student answers from EG and CG to same problem

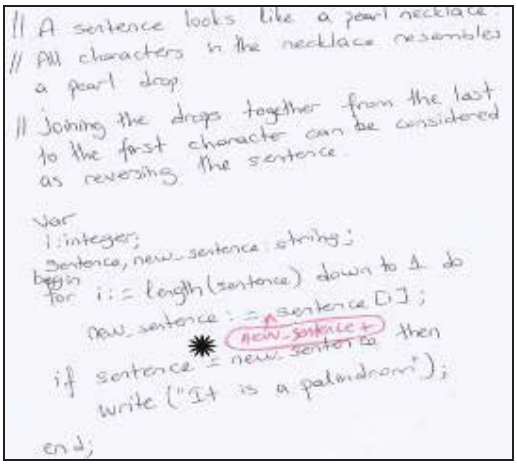
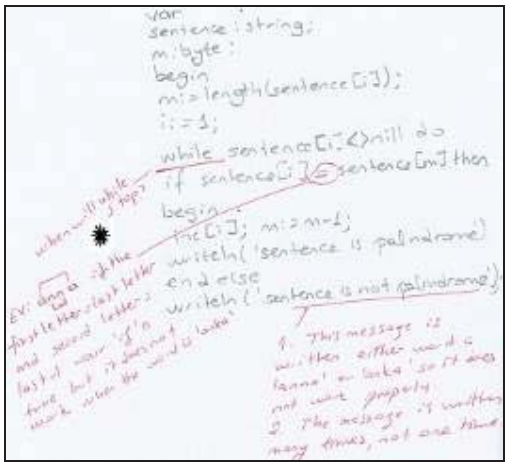
<table>
<tr><td colspan="2">CI with 31(Used CFC and analogy)</td></tr>
<tr><td colspan="2">R: Why did you use pearl necklace?
S31: Pearl necklace has a sequence like the characters of the sentence. If we join these sequences reversely one by one, we will form the reverse of the sentence.
R: Could not you write "k:=k+sentence[i]" without using this analogy.
S31: Honestly, I could. But it would take so time to think the reverse of a sentence. Instructor has told us we can use analogies from daily life in order to develop basic solutions for problems…</td></tr>
<tr><td colspan="2">Q3-3: Input a sentence and print out if it is a palindrome</td></tr>
<tr><td>Student Answer from EG
 
In this answer, the student made an analogy in reversing a sentence. Although it was a good analogy, she made a mistake in joining characters of the sentence. In all steps of the loop she assigned the characters one by one to a memory location (new_sentence). The mistake was that she used the same location for individual characters. After the end of the loop, (new_sentence) includes only the last character of the sentence. Considering this mistake as an iteration problem, it can be evaluated in general programming concept domain.</td><td>Student Answer from CG
 
In this answer, another student from CG has three basic mistakes. First; he did not understand the use of "while" statement that, "while" needs a condition to stop. He could not organize if statement because "else writeln ('sentence is not palindrome')" requires to write message on the form many times. Statement "else" could be provided many times eg. "anka" sentence [1] = sentence [4] ('a'='a') but, sentence[2] <> sentence [3] ('n'<>'k'). Also student understood sentences as array of characters, but could not organize the characters in the string array.</td></tr>
</table>

Table 14: Clinical interviews and student answers (S14)

CI with S14 (Used both approaches)
R: Some of your friends directly begun to write the code, didn't it take so time writing the comments about the problem first? S14: It took much time, of course. But in the class we solve some problems by CFC. By this way, I am writing which way I am thinking. So it helps me on coding…

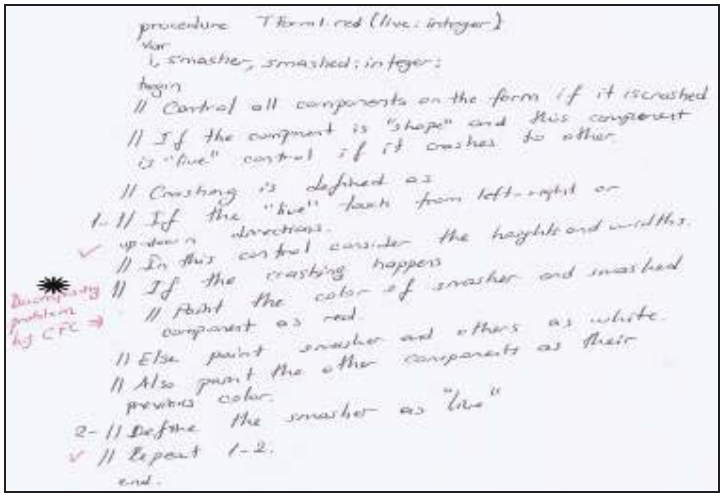
Q4-1: (See Appendix2 (Q4-1)) **Student Answer:** 	In this answer, S14 used CFC in order to divide the problems into sub problems. This was roughly planning the program. He used comment lines before writing codes.

Table 15: Clinical interviews and student answers (S9)

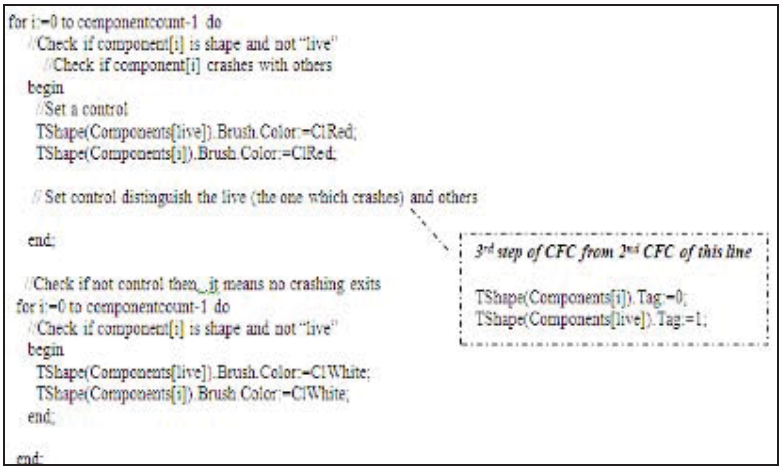
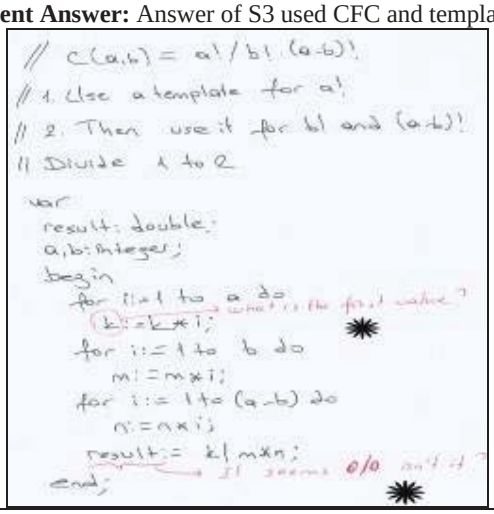
CI with S9(Used only CFC)	
R: Your paper includes more comments, do you really need them? *S9: May be I can code without comments, but sometime I forget what I wrote. So I write comments for whole code to divide it into sub sections. Then I code the sub problems easily.* *R: What is the main advantage using different phase of CFC?* *S10. Sometimes it may help you on solving problems because, you are doing task step by step by CFC...*	
Q4-1: (See Appendix2 (Q4-1)) **Student Answer:** S9 used CFC from second form of CFC. 	The question consists of more than one step for creating shapes, moving the shapes, controlling the crashing and painting the shapes. An example answer used CFC in defining the steps and dividing the problem into sub problems. The student used 3 phases of CFC; mostly the second phase.

Table 16: Clinical interviews and student answers (S3)

CI with S3(Used only CFC and templates)
Student Answers from EG and CG for *R: Why did you use more comment lines?* *S3: I think it is easier to solve the problem by diving the problem into pieces.* *R: What do you mean by the sentence "use a template for a!"* *S3: I mean, I will use some code pieces which I have learnt from the books.*
Q3-2: Compute the combination value of given two integers [C(a,b)]

Student Answer: Answer of S3 used CFC and templates 	S3 from EG used CFC in order to define the program pieces, and also used factorial template in order to calculate the combination. The reviewer noted that student calculated "eg. k:=k*i" but he did not set "k:=1" before "for" loop. The mistake reflects the use of CFC and the factorial template was well thought, but student have problems using the memory functions.

As seen in the examples of student posttest responses, three approaches were used by the students in different forms. All of the three approaches supported EG students in giving more correct answers than CG students. Pretest and posttest comparisons, experts' reviews, clinical interviews and the observations in the EG classroom provided evidence regarding the use of all three approaches with the EG. The expert reviews demonstrated that the students who got higher scores used all of the three approaches intensively. The posttest comparison also provided some evidence of the higher performances of EG students, especially in the problem solving phase. The clinical interviews also showed students' ideas that they generally indicate how they used CFC, analogy and templates.

DISCUSSION

Although there are some research studies which separately investigate the difficulties in students' understanding, these research studies typically used only one teaching approach. This study, using a new approach, presents some evidence of success for novice programmer students, supporting the suggestion that programming language tutors should give attention to the use of appropriate learning activities (Oliver, 1993). Also Ismail, et al (2010) emphasized that programming requires higher level of knowledge at the strategic or conditional level. Thus, CFC and analogy are closely related to conceptual understanding. Because writing the whole code step by step helped students to spend less time on coding, this also encouraged students and they could then start incrementally adding semantics to their programs by including more additional structures. As an example in Table 15, the student put forward the shape movement in his own words, and then he began to adjust the statements systematically. This includes conceptual understanding of the properties of components (shapes). Hence writing correct codes will play an important role for continuing to the next step for students. Sengupta (2009) indicated that CFC improves functionality and reduces the complexity of the program, while allowing the student to compile and test each individual step. This study showed that CFC encouraged students to define the steps of the program and to associate the program's features with students' thoughts, and improved students' coding process.

Concretizing the components such as variables, data types, memory, etc was emphasized as a deficit for students. In this study, students in the EG performed better on overcoming the deficiencies in writing correct codes, which are related to the conceptual understanding. Thus, the basic programming concepts which are taught by analogy are also positively affected by conceptual understanding. By using analogy; memory, disk allocations, files options are well understood and used by students. Posttests showed that, analogies were also used in problem solving. The "pearl necklace" was an interesting example in enhancing students' cognitive development by providing scaffolding for them to construct new knowledge based on their previous knowledge. Since analogy is a complex cognitive process, this study can be considered as an example so that efforts in other domains may be adapted to programming concepts. Consequently, the proposed approach for teaching introductory programming, that is, analogy, may be a tool for teaching abstract concepts in programming.

The template approach can be considered to be a facilitator for problem solving. In this study, templates which include the procedural skills of planning, testing and reformulating codes were improved the students' achievements in problem solving phase. A swapping template used by students was a simple example for using a swapping template in sorting, merging or other applications; it means reusing the template. Schank, Linn & Clancy (1993) suggested that using the templates supported students in remembering and reusing information,

and students may gain a deeper understanding if the representation was introduced in the context of a programming case study. In this study, students planning with available templates reduced the task because some of the smaller programs were already written. From a theoretical viewpoint some complex subjects can be understood easily by looking at the previous related subjects (Dunbar, 2001). So by using templates students can use expert strategies (including the use of the procedural skills) abstracted from the specific language.

The results showed that, in the conceptual understanding domain, both the EG and CG had nearly equal performances that were statistically insignificant. This may be related to the structure of the entry-level programming concepts presented to students. In this domain, the average of EG posttests is greater than CG, which might indicate a positive effect of the hybrid approach in understanding language features and general programming concepts. Also the clinical interviews and student posttest responses showed that the EG students made mistakes in using memory, iterations, variables and arrays less frequently than the CG. This can be interpreted as the analogies playing a facilitator role; but this does not mean that the EG students understood the concepts better than the CG. In the problem solving domain, there was a significant difference between the groups. Especially in Q4 which measures analytical thinking, programming language structure knowledge, and correct code writing skills; the difference between groups was prominent in favor of the EG. The students in the EG could use the templates in order to write the "crashing and painting the components" in this question. Also using CFC, they eliminated the syntax errors and this motivated the students to continue on in their projects. In contrast to the EG, some students left the projects as a result of uncorrected syntax errors in the CG. So even if the syntax errors did not directly affect solving the problems, overcoming this problem facilitated continuation of problem solving. As Rist (1991) highlights, the main source of difficulty does not seem to be only in the syntax or understanding of concepts, but rather in the problem solving. A student can learn the meaning of array, but still can fail to use it appropriately in problem solving. In this study analogies helped students in drawing an analogy between possible solutions, so students in EG could decompose the program, and solve them more easily. Templates also facilitated problem solving, because students used the templates as soon as possible to reach a code difficult to write, which allowed them to integrate their own programs.

A number of previous studies have demonstrated that instruction using these approaches solely also improve learners' conceptual knowledge of programming (Robins, Rountree & Rountree, 2003; Sengupta, 2009). This study, using a new approach, found that the programming knowledge of novices significantly improved when appropriate training was provided through the combination of three approaches. This may suggest that using the approaches in combination of one and the other may increase performance better than using one approach in solely. Related to Table 12, we can have an idea about the effects of approaches that; CFC and analogies together were affected on conceptual understanding more and also CFC, analogies and templates together were effective on problem solving domains. Some other experimental studies are needed for determining the absolute effects of the three approaches.

In addition; the study has some limitations. During the lessons in CG, the use of CFC and templates were kept entirely under control. But instructor has used a limited number of analogies while providing natural explanations for programming codes. Instructor used them necessarily, because without using them, it would be difficult for students to understand the subjects to which the analogies pertained. But the analogies used were limited in number and not detailed like the ones employed in EG. It was very difficult to avoid this situation in CG so it may be considered to be a limitation for this study. Another limitation is the number of participants in the groups. In fact, in both groups there were more students than those included in the study. Some limitations have led to some of the students not to be included in this study. In this context, students whose pretest and post test scores were very low (less than 5/100) were not included in the study. Also, in both groups, students who had more absences than was allowed were not included in the study. In addition, during the process of EG; in the classroom, they worked with the templates which the instructor has chosen. He only guided for them about how they can find other templates about related subjects.

CONCLUSION AND IMPLICATIONS

Researchers are still trying out new approaches for teaching programming. In this study, students who used the hybrid approach significantly outperformed the students who followed the traditional programming course. The results of this study suggest that combining different approaches may enhance programming tutoring and this may have positive effects on students' development of their introductory programming knowledge, so it may be used as a facilitator. The hybrid approach provided an opportunity for students to use their cognitive strategies and relevant conceptual knowledge in programming languages contexts. By using the hybrid approach, instructors began to explain why and how programs work, what the strategies are for decomposing tasks or problem solving, rules for the creation of well-formed programs and design features. It also provided experience of using the CFC method in the classroom for some students who could not produce correct pseudo codes, or

could not transfer their pseudo codes into a true code. Thus the results showed that CFC may have an effect on applying the correct rules of syntax when programming, and analogies on the development phase of programs, while templates enhance communication for getting support from sample codes to solve problems. Despite many advantages of the new approach, special efforts must be made to correct some mistakes, especially with regard to iteration, variable usage and memory operations.

In addition instructors should know that learning programming is a complex process. In order to facilitate this process; they can use CFC, analogies or templates. The main issue regarding the use of these approaches is answering the basic question; "Where and when should the instructor use these approaches?" According to the results of this study; instructors may use analogies when teaching abstract subjects like memory and disk allocations. They can use templates, especially in problem solving, to refer the good solutions. CFC can be used in the first phase of code writing, it helps students to ensure about their code syntax are true. Also decomposing the problems with CFC will encourage students to continue on writing. This is also important because some students give up writing because of syntax errors. The results of this study showed that students with higher scores used two or more approaches together in their solutions. Therefore instructors may use the approaches together when they solve problems depending on the nature of the problem. Of course it is impossible to apply the approaches at every stage of the programming lessons. A proper planning is therefore needed to adopt these approaches to the lessons. For this reason programming instructors should search the templates and should think about analogies before lecturing.

The results have two important implications: Firstly; programming course is not a fearful course. Learning to program can be improved through training via convenient approaches. Secondly; by learning new approaches in the field, instructors can find new solutions to deficiencies. Finally; this study illustrates that a positive effect is derived through the multiple usage of the three approaches. However, there is a need for conducting more clinical interviews and gathering instructors' perspectives in detail to identify how effective each approach is. In future research, other data collection tools may be used to determine in depth programming process issues which are not addressed properly; for example, efficiency, meaning, purpose, proper usage of codes, and so on.

REFERENCES

Barg, M., Fekete, A., Greening, T., Hollands, O., Kay, J., Kingston, J. H. & Crawford, K. (2000). Problem-based learning for foundation computer science courses. *Computer* Science *Education, 10* (2), 109-128.

Byckling, P. & Sajaniemi, J. (2006). Roles of variables and programming skills improvement. *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2006)*, 413–417.

Bayman, P. & Mayer, R.E. (1988). Using conceptual models to teach basic computer programming. *Journal of Educational Psychology*, 80, 291-298.

Carbone, A. & Sheard, J. (2002). A studio-based teaching and learning model in it: what do first year students think? *Seventh Annual Conference on Innovation and Technology in Computer Science Education*, University of Aarhus, Denmark.

De Raadt, M. (2007). A review of Australasian investigations into problem solving and the novice programmer. *Computer Science Education, 17* (3), 201 – 213.

Dunbar, K. (2001). The analogical paradox: Why analogy is so easy in naturalistic settings, yet so difficult in the psychology laboratory. In D. Gentner, K. J. Holyoak, & B. Kokinov (Eds.), Analogy: Perspectives from Cognitive Science. Cambridge, MA: MIT .

Dunican, E. (2002). Making the analogy: Alternative delivery techniques for first year programming courses. *Proceedings from the 14th Workshop of the Psychology of Programming Interest Group,* Brunel University, 89-99.

Gentner, D. (2000). Perspectives from Cognitive Science. MIT press Cambridge MA.

Govender, I. & Grayson, D. (2006). Learning to program and learning to teach programming: A closer look. In E. Pearson & P. Bohman (Eds.), *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2006* (pp. 1687-1693). Chesapeake, VA: AACE.

Hagan, D.L. & Macdonald, I.D.H. (2000). A collaborative project to improve teaching and learning in first year programming. *Australasian Journal of Engineering* Education. *9* (1), 65-76.

Hui Hui, T. & Umar, I,N. (2011) Does a combination of metaphor and pairing activity help programming performance of students with different self regulated learning level? *The Turkish Online Journal of Educational Technology, 10* (4), 121-129

Ismail, M.N., Ngah, N.A. & Umar, I.N. (2010). Instructional strategy in the teaching of computer programming: a need assessment analyses. *The Turkish Online Journal of Educational Technology, 9* (2), 125-131.

Jenkins,T.(2002).On the difficulty of learning to program. http://www.ics.ltsn.ac.uk//pub/conf2002/ jenkins.html, Accessed January, 2011.

Linn, M., & Dalbey, J. (1989). Cognitive consequences of programming instruction. In E. Soloway & J. C. Spohrer (Eds.), Studying the novice programmer (pp. 57-81). Hillsdale, NJ: Lawrence Erlbaum.

Ma., L., Ferguson, M. R & Wood, M. (2011) Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education, 21*(1), 57–80.

Mannila, L. (2007). Novices' Progress in Introductory Programming Courses. *Informatics in Education, 6* (1), 139–152.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B.-D., Laxer, C., Thomas,L., Utting, I. & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin, 33* (4), 125-140.

McGill, T.J. & Volet, S.E. (1997). A conceptual framework for analysing students' knowledge of programming. *Journal of Research on Computing in Education, 29* (3), 276-297.

Miliszewska, I. & Tan, G. (2007). Befriending computer programming: a proposed approach to teaching introductory programming, *Issues in Informing Science and Information Technology*, 4, 277-289.

Oliver, R. (1993). The contextual model: An alternative model for teaching introductory computer programming. *Journal of Computers in Mathematics and Science Education, 12* (2), 147-167.

Proulx, V. (2000). Programming patterns and design patterns in the introductory computer science course. *SIGCSE Bulletin, 32* (1), 80-84.

Robins, A., Rountree, J. & Rountree, N. (2003). Learning and teaching programming: a review and discussion. *Computer Science Education, 13*, 137 -173.

Rist, R. (1996). Teaching Eiffel as a first language. *Journal of Object-Oriented Programming, 9*, 30-41.

Schank,P.K., Linn,M., C. & Clancy, M.J. (1993). Supporting Pascal programming with an on-line template library and case studies. *International Journal of Man-Machine Studies, 38*, 1031-1048.

Sengupta, A (2009). CFC (comment-first-coding) – a simple yet effective method for teaching programming to information systems students. *Journal of Information Systems Education, 18*, 1.

Siegle, D. (2009). Developing student programming and problem-solving skills with visual basic. *Gifted Child Today, 32*, 24-29.

Stein, L. A. (1999). Challenging the computational metaphor: implications for how we think. *Cybernetics and Systems, 30* (6), 1-35.

Stemler, S. E. (2004). A comparison of consensus, consistency, and measurement approaches to estimating interrater reliability. *Practical Assessment, Research & Evaluation, 9* (4).

Vujosevic-Janici, M. & Tosi'c, D. (2008). The role of programming paradigms in the first programming courses. *The Teaching of Mathematics, 9* (2), 63–83.

Weigend, M. (2006). From intuition to programme. Programming versus application. In:Mittermeir, R.T. (Ed.), *ISSEP 2006*, LNCS, 4226, 117–126.

Wiedenbeck, S. & Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human-Computer Studies, 51*, 71-87.

Williams, L. A. & Kessler, R. R. (2000). Effects of 'pair-pressure' and 'pair-learning' on software engineering education, *The 13th Conference on Software Engineering Education and Conference, IEEE Computer Society*, Austin, TX, 59 - 65.

Winslow, L.E. (1996). Programming pedagogy – A psychological overview. *SIGCSE Bulletin, 28*, 17–22.

Thuné, M. & Eckerdal,A. (2009). Variation theory applied to students' conceptions of computer programming. *European Journal of Engineering Education, 34* (4), 339-347.

Appendix 1. Pretest questions for determining the programming knowledge

Language Features	1. Fill in the blanks according to programming language code.
	Uses
	crt;
	var
	a,b,c,max:_____;
	begin
	clrscr;
	write('1.number:');readln(a);
	max:=a;
	_____('2.number:');readln(b);
	if b>max then
	max:=b;
	write('3.number:');readln(c)
	if c>max then
	_____:=c;
	write('Maximum:',_____);
	readln;
	end.

Basic Programming Concepts	2. Please define a)Variable, constant b)Memory, file c)Matrix
Simple Problem	3. Write a code (use any programming language you know) to solve the problem. A tree is X meter long and extends up to 2% of the length of each year. How many meters in length will it be Y years later?

Appendix 2. Post Test (ProKT) Questions for determining programming knowledge

Phase	Number	Question
Conceptual understanding	**Q1**	**During program coding why do you use (16 p)** 1.Variables with meaningful names 2.Functions 3.Iterations 4.Commenting
	Q2	**Find the first 200 Fibonacci numbers (24 p)** 1.By using functions 2.By using procedures 3.Use none of functions of procedures 4.Explain the differences between procedure and a function 5.Explain the life cycle of parameters in your function and procedures 6.How does your program activate a procedure and a function
Problem solving via programming languages	**Q3**	**Write a code (24 p)** 1.Compute the sort of given numbers 2.Compute the combination value of given two integers (C(a,b)) 3. Input a sentence and print out if it is a palindrome *Input Sentence*: "*Madam I'm Adam*" *Output* "It is a palindrome"
	Q4	**Develop a program (36 p)** 1. Develop a program about the movements of a limited number of shapes in the Delphi form. Paint all the shapes in same color, and move the shapes by pressing direction buttons. If live shape crashes other one, paint them in red color.
You should explain why you thought to use your way in all of the Questions.		

Appendix 3. Post Test Grading Table

Question	Criteria	Point
Q1.1, Q1.2, Q1.3, Q1.4	Making meaningful explanations.	6
	Giving examples.	5
	Using the correct terminology in explanations.	5
Q2.1	Writing functions correctly (syntax and structure).	4
Q2.2	Writing correct procedure (syntax and structure).	4
Q2.3	Provide correct solution and writing correct code.	4
Q2.4	Meaningful explanation about difference.	4
Q2.5	Authentic explanation and giving example	4
Q2.6	Meaningful explanation on the given code	4
Q3.1, Q3.2 Q3.3	Using true syntax.	5
	Using code editor correct.	5
	Small size of code.	5
	Effectiveness of algorithm.	5
	Generating true results.	5
Q4	The criteria in 3rd question and also the below	25
	Using general programming structures.	5
	Using the best feasible structures.	5