International Journal of Computer Applications (0975 – 8887)
Volume 100– No.4, August 2014

Learning Programming: A Model Emerging from Data

Nazir Hawi, Ph.D
Department of Computer Science
Faculty of Natural and Applied Sciences
Notre Dame University - Louaize

ABSTRACT

Learning computer programming is a prominent issue in the fields of computer science and education. This paper is an attempt to address this issue by investigating the experiences of undergraduate university students who studied computer programming. A total of 260 computer science and engineering students (210 males and 50 females) were recruited from three geographically distant campuses. They were surveyed with a questionnaire that exhibited good internal reliability. Eventually, a learning model emerged from the data. It consisted of three independent structures that included most of the study times under focus. The study times creatively and dynamically interact stimulated by learning needs and sustained by learning passion. In addition, while some indices of confirmatory factor analysis indicated that the learning model is an adequate fit, others suggested that it needs improvement, which should be considered by future research.

Keywords

Learning computer programming, Higher education, Educational environment

1. INTRODUCTION

Many scholars have acknowledged that computer programming is not easy for many students [1-8]. Despite the wealth of general learning theories and research that addressed learning computer programming in particular [9, 10], pessimism remains prevalent about their effects on classroom practice [11, 12]. Specifically, researchers have been investigating factors and strategies that make learning computer programming successful [13-16]. Indeed, evidence supports that using inappropriate learning strategies leads students to struggle, fail, or drop out [17-19] despite the presence of ability and/or exercise of effort [15, 18]. The aforementioned concerns higher education educators from computer science, engineering, management information [17], and some business tracks [19]. In addition, it concerns researchers who are trying to develop new processes for the learning of software developers in industrial settings [20, 21]. The multidisciplinary nature of this concern makes its investigation very difficult because its basic requirement is researchers that are simultaneously proficient in at least two disciplines, computer science, and education.

2. LITERATURE REVIEW

The widespread conviction that computer programming is a difficult subject triggered the interest of many to understand the underlying reasons and search for remedies. Many educators have been raising the following question: what makes students succeed in computer programming courses? [1, 16]. This question is mainly posed to help learners who experience difficulties in learning this topic to continue to progress. To this end, researchers have been studying the constituents of the computer programming educational environment: the subject *per se*, its learners, its teaching and learning methods, and instructional technology.

Regarding the subject, studies that focused on identifying difficult concepts found many [5, 11, 22-24]. Concerning learners, studies showed that prior computer experience did not affect achievement in computer programming [1, 13, 25-27]. Some of the research that focused on learners of computer programming produced conflicting results. First, while [14] found that learning styles did not influence achievement, [8] stated that learning styles enhance the learning experience because they provide opportunities for all types of learners. Moreover, while some studies revealed that the mathematics background significantly influenced performance [13, 27], others showed it had no such effect [1]. Furthermore, studies that identified the best predictor of success in programming courses yielded incongruent results. The best predictor was comfort level for [27], positive attitude for [26], and students’ perception of their understanding of the module for [13]. The latter predictor finds some support in [27] study that identified attribution for success/failure as a predictive factor. Furthermore, [19] stated that attribution to luck negatively affected performance. These conflicting results are probably due to varying individual factors with and within educational environments. In fact, to a given university computer programming class, students bring with them diverse mathematics abilities and background knowledge.

[14] research results indicated that self-regulatory learning plays a role in learning to program. Nevertheless, other traits are needed at work. [20] observed that many of the problems newly hired developers experienced resulted from poor communication skills and social naivety. They suggested the use of particular effective instructional pedagogy, such as pair programming, to better prepare students to fit technically and socially in the industry. In their study, pair programming increased student performance, self-efficacy, and confidence in asking questions. Furthermore, [3] showed that pair programming is an effective technique for helping students learn to program. Pair programming sounds promising because its proper implementation may immediately bring about desired outcome.

With respect to instructional technology, [9] recommended the replacement of a straight lecture format with active-learning exercises that utilize technology. Active learning made participants significantly more attentive. Furthermore, [28] posited that hands-on practice in class is a promising instructional approach. Moreover, [29] stated that “students migrated from the state of passive receivers to constructors of computer programming concepts” due to the use of technology in class and learner-centered approaches. In this educational environment, in-class activities that employ a software development tool and a projector allow for natural collaboration, including peer programing. In addition, [30] reported that the inclusion of open-ended extensions in assignments offers opportunities for optional extra-credit work that can take interested learners beyond the requirement. Moreover, [30] conducted optional programming contests as a strategy to capture the interest of the most highly motivated students. In addition, [2] presented the following three techniques: worked examples, example completion, and variability. [17] proposed the use of debugging and modifying programs to develop students' programming mental model in order to improve their performance. In addition, they suggested using frequent assignments with quick and ample feedback rather than a smaller number of long-term projects. Furthermore, they stated that peer modeling, group work, strong interaction with other students, and a calm physiological state can promote self-efficacy. Other scholars reached the same conclusion [31].

[3] noted that when programming students get stuck, they frequently search the textbook for solutions. [6] reported that their subjects read, looked information up on the Internet, and used tools. Subjects sought help from peers, their instructor, or other faculty members. Furthermore, students practiced, learned from examples, and invented tasks beyond the assigned work. In addition, research that focused on how students study computer programming courses presented suggestions to simplify studying this subject [23]. [10] provided a handout of representative advice, including tinkering, working incrementally, and conceptual understanding. [7] suggested using pattern tutorials developed specifically for students to learn essential thinking skills needed in introductory programming. Other researchers focused on the concept of a learning strategy. The learning strategy is the learner's actual study approach to achieve the computer programming learning objectives using topic-appropriate study skills. This definition begs the following question: can a particular computer programming learning strategy ensure success? [16] showed that good performance in object-oriented programming is strongly associated with a strategic learning approach. All of the aforementioned research shows that a learning model that encompasses the major constituents of a computer programming learning strategy is urgently needed. Such a learning model may serve as a bridge between research and practice.

2.1 Problem statement

Some scholars agree that learning computer programming is difficult and complex for students to learn. Consequently, many have attempted to obtain solutions to learning challenges of computer programming, but most of these solutions were observation-based research from the researchers' perspective. Research that considers the students' voice, as well as its scientific analysis and synthesis, is scarce. This research intends to fill the gap by capitalizing on the personal experiences of students [1, 6, 10] who completed at least one computer programming course at the undergraduate level, prior research results, and the experiences of higher education experts in the field. These insights were thoroughly investigated to understand how students learn computer programming in order to design a computer programming learning model that can be evaluated and developed as well as facilitate the education of those who utilize it.

2.2 Research questions

1. Is it possible to identify a computer programming learning model?
2. If such a model exists, what are its basic constituents and how do they interact?

3. METHOD

3.1 Participants

The study was conducted on three campuses at Notre Dame University - Louaize, Lebanon, in the Faculty of Natural and Applied Sciences (FNAS) and the Faculty of Engineering (FE). The Faculty of Natural and Applied Sciences (FNAS) participants were from the Computer Science Department majoring in Computer Science (CS), Computer Information Systems (CIS), Computer Graphics and Animation (CGA), Business Computing (BC), and Geographic Information Systems (GIS). The Faculty of Engineering (FE) participants were from three departments Mechanical Engineering majoring in Mechanical Engineering (ME), Electrical and Computer and Communication Engineering majoring in Electrical Engineering (EE) and Computer and Communication Engineering (CCE), and Civil and Environmental Engineering majoring in Civil Engineering (CE).

Undergraduate students enrolled in these two faculties who completed at least one computer programming course qualified for the survey. A total of 260 students (80.8% males and 19.2% females) participated in the study forming more than a quarter (27.8%) of FNAS and FE. Fifteen questionnaires were unusable. The age characteristics of the remaining 245 participants (M = 21.4, SD = 2.1), aged 18-40 years, were very close to those of the target population (M = 21.0, SD = 1.8), aged 17-40 years. The majority of participants were sophomores and juniors.

3.2 Computer programming courses

Table 1 lists the computer programming courses offered by the computer science department that FNAS/FE participants have mentioned in relation to their learning experience. Each row represents one of these courses and shows the course number and title, the used development language, the tracks that need it, and the prerequisite course if any. Languages were chosen by faculty based on academic tracks, industry relevance, and market demand [12]. For instance, all Engineering, Computer Science, and Computer Information Systems students take the introductory computer programming course CSC 212.

All computer programming courses are 3-credit courses. One credit is equivalent to 1-hour in time. A computer programming course is completed over a 15-week semester. In fall and spring semesters, a course corresponds to 45 hours of in-class work. Although summer semesters extend over six weeks, the number of in-class contact hours remains the same. Forty-five hours per course expose students to learning experiences sufficient to enrich knowledge on how they study for computer programming.

At the computer science department, there are three sequences of computer programming courses (CSC 212, CSC 213, and CSC 313; CSC 216, CSC 217, and CSC 417; CSC 220, CSC 320, and CSC 387). Introductory and intermediate level computer programming courses have many topics in common. For instance, of the common topics of all introductory level courses the list includes variables, assignment statements, input and output, arithmetic operations, logic operators, control structures, functions, subs, arrays, and object-oriented programming. An example of a topic taught in CSC 216, but not in CSC 212, is event-driven programming.

3.3 The survey questionnaire

The starting point was to create a survey questionnaire whose aim was to investigate undergraduates' personal experiences in learning computer programming. This process entailed reviewing prior research articles that investigated learning computer programming to ensure the capture of all relevant information. A draft survey questionnaire was developed based on all of these reviews and the researcher's 27 years of teaching computer programming. Five expert faculty members in the field then studied the draft questionnaire. Furthermore, the input of students who completed computer programming courses was sought. From the experts, the students, and the researcher, the survey questionnaire emerged with six different constructs: reading, practice, help, classroom activities, problem solving, and complementary programming skills.

Table 1. Programming courses where participants had their learning experiences

Number	Title	Language	Tracks	Prerequisite
CSC 212	Program Design & Data Abstraction I	C++	CS; CIS; ME; CCE; CE; EE	None
CSC 213	Program Design & Data Abstraction II	C++	CS; CIS; CCE; EE	CSC 212
CSC 216	Computer Programming I	VB.NET	BC; GIS	None
CSC 217	Computer Programming II	VB.NET	BC; GIS	CSC 216
CSC 220	Programming in Java I	Java	CGA	None
CSC 313	Data Structures using C++	C++	CS; CIS	CSC 213
CSC 320	Programming in Java II	Java	CGA	CSC 220
CSC 387	Advanced Programming using Java	Java	CGA	CSC 320
CSC 417	Advanced Programming Technologies	VB.NET	BC; GIS	CSC 217

Note. CS**:** Computer Science; CIS: Computer Information Systems; CGA: Computer Graphics and Animation; BC: Business Computing; GIS: Geographic Information Systems; ME: Mechanical Engineering;
EE: Electrical Engineering; CCE: Computer and Communication Engineering; and CE: Civil Engineering.

Each construct was broken down into a set of specific measures. Overall, the survey consisted of eight different sections that encompassed 29 items. Despite all attempts, the draft survey questionnaire could not have been shorter. The first section consisted of demographic questions (age, gender, computer programming course, and grade). The second section included three measures of the reading construct and one open-ended item. Except for the latter, the responses were marked on a three-point scale: *daily*, *occasionally*, and *never*. The expert faculty suggested the three-point scale to ensure that possible responses are mutually exclusive. Furthermore, they recommended the words *daily*, *occasionally*, and *never* because they are consistent with the terminology employed by teachers and students. The last section included one open-ended question that gave participants the chance to specify how their learning strategy could have been better. In fact, the last item of each construct was open-ended. These items were considered useful to identify new information that was not accounted for in the survey questionnaire. Because the survey instrument was newly constructed for this study, its internal reliability was checked using Cronbach's alpha (α), the most popular coefficient of reliability measure [32]. Cronbach's alpha (.770) exceeded the threshold of .700, indicating that the instrument's items had good internal reliability.

3.4 Data collection

Permission to administer the survey questionnaire was obtained from the deans, chairpersons, and instructors, in that order. All aforementioned people gave their consent, expressed their belief that the study was important, and asked to see results. The survey questionnaire was explained to students in regular classrooms, including its purpose and the expected 15-min completion time. In addition, they were told that their participation was voluntary. Completion of the questionnaire was anonymous with a promise of confidentiality. The meaning of analytical thinking was explained to some students upon their request. After submission, students and instructors were thanked for their participation.

3.5 Research design

The data for this study were collected using the survey questionnaire. The quantitative data were processed under descriptive and inferential statistics using SPSS 20 procedures. Furthermore, a confirmatory factor analysis was performed in structural equation modeling with IBM SPSS Amos Graphics 20 to test the structure underlying the set of 20 study times. The qualitative data were processed under content analysis using HyperRESEARCH 3.0 that categorized the students' replies to open-ended items.

4. RESULTS

4.1 Results regarding learning strategy

4.1.1 Analysis of Study Time with Three Categories for Response Using Chi-Square Test

The chi-square test was computed for items that provided three categories for a response, *daily*, *occasionally*, or *never* (see Table 2). The aim was to determine the count of students who daily, occasionally, and never study. The predictive hypothesis was that at least one of the categories exceeded the other two. In Table 2, the degree of freedom was 2. Furthermore, the asymptotic significance was less than .001; thus, the change was significant because the data set was large (245). All rows had an expected count of less than five. The proportions were unequal for all items. For instance, students preferred one or more of the item '*Reading the course's assigned textbooks*' categories, with $\chi^2(2, N = 251) = 88$ and $p = .005$. Table 2 indicates that the category *occasionally* was the most preferred by students. Overall, more than four quarters (76.9%) of the students read the course's assigned textbook with 99% CI [70.0, 83.7].

Studying occasionally was the most prevalent pattern of carrying out every study activity without exception. Future studies should examine whether this behavior is sociocultural.

Table 2. ***Chi-Square Test for Research Variables under Reading, Practice, Help, and Classroom***

Study Time	**Daily**	**Occasionally**	**Never**	**Chi-Square**	**Usable % D & O**[*]	**99% Confidence Interval**	
Reading							
Reading the course's assigned textbooks (R1)	40	153	58	88	76.9	.700	.837
Reading additional textbooks (R2)	6	66	179	184	28.7	.213	.360
Reading online resources (R3)	24	123	104	66	58.6	.506	.666
Practice							
Practice by solving problems from the textbook(s) (Pra1)	36	151	64	86	74.5	.674	.816
Practice by developing code blocks or applications for novel situations (Pra2)	29	124	93	57	62.2	.542	.702
Help							
Seeking help from my instructor (H1)	60	162	29	116	88.4	.833	.936
Seeking help from computer programmers (H2)	13	100	138	98	45.0	.369	.531
Seeking help from programming language news groups (H3)	10	56	185	197	26.3	.191	.335
Classroom							
Participation (Cl1)	150	86	12	115	95.2	.917	.987
In-class understanding of computer programming concepts (Cl2)	178	64	7	323	97.2	.945	.999

Note. The chi-square values are rounded to the nearest whole number.
*The usable percentage of participants who responded daily and occasionally.

Determining the causes may lead to remedies. Second, the *classroom* study time contained the largest percentages for the combined *daily* (95.2%) and *occasionally* (97.2%) categories. The extreme importance students attach to classroom variables must be cultivated by devising learning activities that maximize in-class learning and instigate the interest in students to extend their learning experience outside the classroom. Combining the first and second findings identifies a sizable group of learners who regularly participate in class discussions and activities by choice to understand subjects but irregularly carry out relevant after-class activities. Third, *help, practice*, and *reading* had high percentages for their study times '*Seeking help from my instructor*' (88.4%), '*Practice by solving problems from the textbook(s)*' (74.5%), and '*Reading the course's assigned textbooks'* (76.9%), respectively. However, the remaining percentages require attention, as they might reveal serious cases of reluctance due to fear or embarrassment or struggle due to misconception or concealing that require intervention and remedy. Fourth, the reading study time shows that reading online resources (58.6%) is more common than reading additional textbooks (28.7%). Consequently, adding links to useful sites from the instructor's perspective or interesting sites from the learner's perspective [33] that are relevant to the course content to the syllabus or the course's account on an e-learning system is valuable for fueling motivation. Nevertheless, the aversion to reading additional textbooks is alarming as it hinders the acquisition of general knowledge and thus requires further investigation into its causes and effects.

Although this model appears applicable across disciplines, the computer programming learning process must occur in a synchronistic manner. For instance, a learner can read, practice, and seek help in that order. A learner cannot only read throughout the course or only practice [34]. Synchronistic learning is problematic for many learners, especially when 1) they are novices studying such a demanding subject, 2) they are not proficient at one or more of the aforementioned studying skills, and 3) they are not taught and guided appropriately.

4.1.2 Analysis of Study Time with Binomial Categories for Response Using Percentages

Table 3 shows that respondents attached the highest importance in problem-solving to '*Analytical thinking*' and '*Learning how to convert the problem solution to code*', in that order. In fact, all problem-solving study times were deemed important by a high percentage of respondents. The complementary programming skill study times were not deemed as important as those of problem solving.

4.1.3 Spearman's Rho Correlation

Spearman's rank correlation coefficient (*rho*) was used to explore the relationships among the twenty study times. The exploration did not show any negative correlations. In addition, the analysis showed many medium and few large correlations. The three self-contained clusters formed by these correlations supersede the intrinsic importance of these correlations (see Table 4). Figure 1 visually illustrates these clusters and few sub-clusters. The set of clusters will be examined as a learning model encompassing three learning structures. A learning structure of matching study time corresponds to each cluster formed by a set of correlations. Due to their weak correlations, the study times '*Reading additional textbooks*' (R2), '*Learning to test my code*' (Pro7), and '*Improving my typing*' (Co3) were not part of the model. A double-headed curvilinear arrow represents a correlation between two study times. For instance, in-class '*Participation*' (Cl1) directly correlates with after-class '*Seeking help from my instructor*' (H1).

Table 3. ***Study times with binomial categories for response***

Study Time	**Yes (%)**	**No (%)**
Problem Solving		
Analytical thinking (Pro1)	94.4	5.6
Learning how to convert the problem solution to code (Pro2)	94.0	6.0
Learning to locate and solve syntactical errors (Pro3)	82.4	17.6
Learning to locate and solve logical errors (Pro4)	85.1	14.9
Learning to locate and solve run-time errors (Pro5)	75.9	24.1
Learning about and using objects, methods and functions (Pro6)	90.8	9.2
Learning to test my code (Pro7)	90.4	9.6
Complementary programming skills		
Learning how to use the development environment (Co1)	56.0	44.0
Learning how to use the development environment's editor (Co2)	40.5	59.5
Improving my typing (Co3)	69.6	30.4

Table 4. Correlations of study times

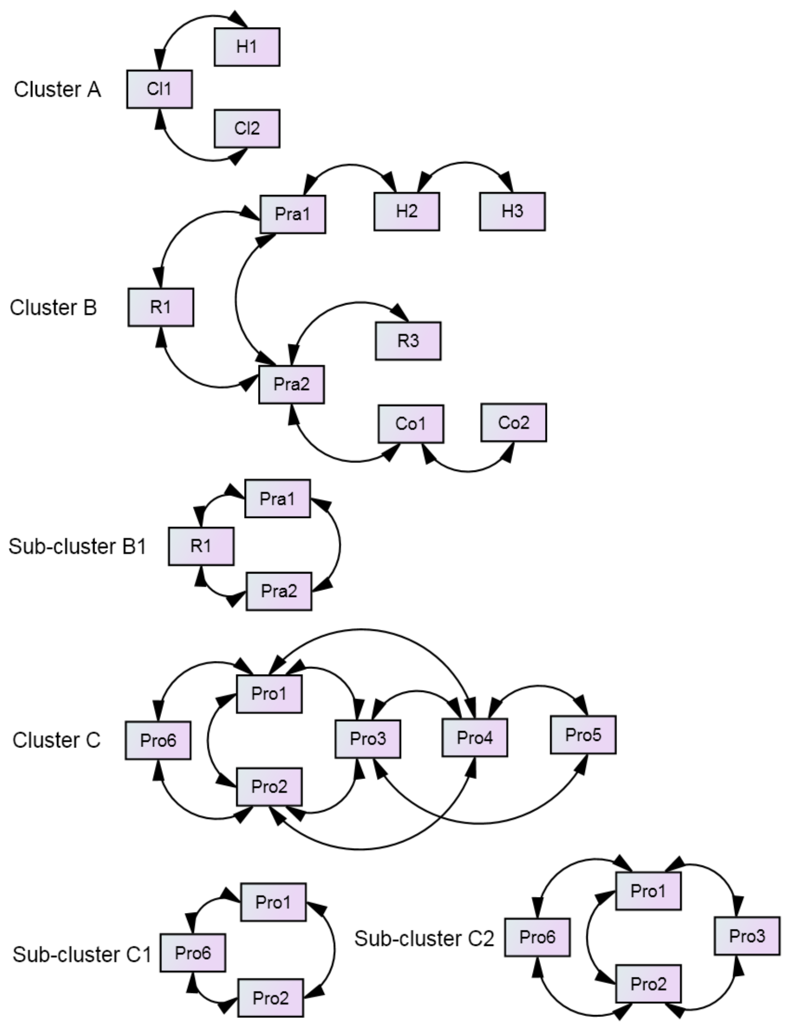
Variable	***R1***	***R3***	***Pra1***	***H2***	***Cl1***	***Pro1***	***Pro2***	***Pro4***	***Pro5***	***Co1***
Pra1	.329			.307						
Pra2	.306	.341	.338							.333
H1					.312					
H3				.321						
Cl2					.457					
Pro1							.527			
Pro3						.311	.301	.466	.405	
Pro4						.301	.300		.398	
Pro6						.535	.358			
Co2										.660

Note. R1: Reading the course's assigned textbooks; R3: Reading online resources; Pra1: Practice by solving problems from the textbook(s); Pra2: Practice by developing code blocks or applications for novel situations; H1: Seeking help from my instructor; H2: Seeking help from computer programmers; H3: Seeking help from programming language news groups; Cl1: Participation; Cl2: In-class understanding of computer programming concepts; Pro1: Analytical thinking; Pro2: Learning how to convert the problem solution to code; Pro3: Learning to locate and solve syntactical errors; Pro4: Learning to locate and solve logical errors; Pro5: Learning to locate and solve run-time errors; Pro6: Learning about and using objects, methods and functions; Co1: Learning how to use the development environment; Co2: Learning how to use the development environment's editor.
*All values were significant at the .001 level (2-tailed).

Cluster A is the simplest learning structure. It is a two-level mapping of study times (see Figure 1). At level 1, this cluster has one core in-class '*Participation*' (Cl1). Participation is the artifact of students' sociocultural identity mediated by the instructor's teaching-learning model. In turn, participation correlates with two independent study times '*Seeking help from my instructor*' (H1) and '*In-class understanding of computer programming concepts*' (Cl2). While '*Participation*' (Cl1) and '*Seeking help from my instructor*' (H1) are observed behaviors that are manifestations of mental processes, such as thinking, '*In-class understanding of computer programming concepts*' (Cl2) is a mental activity. The three study times are employed to meet individualized needs during the physical presence of the course instructor, whether in the classroom, office, or hallway. Cluster A is independent of the other two clusters. Its study time does not link to the study time of other clusters, i.e. students who engage in in-class participation do not necessarily engage in after-class reading and/or practice. Experience supports this phenomenon; some students depend solely on classroom activities. Students who have a learning approach that does not employ a learning structure other than that of Cluster A may never be autonomous learners, which hinders lifelong learning [35]. [18] stated, efforts must be invested to identify and turn ineffective novices with excessive dependence on assistance to effective learners. This cluster presents a learning structure that is relevant to fields of study other than computer programming.

Cluster B is more complex than Cluster A. It is a four-level mapping of study times that is independent of the other two clusters. Cluster B's core is reading, specifically '*Reading the course's assigned textbooks'* (R1). Thus, the choice of one or more appropriate programming course textbooks is essential to learning [36]. Reading an appropriate textbook correlates with practice [36]. As mentioned earlier, [3] noted that programming students frequently search the textbook for solutions when they encounter problems during programming sessions. At level 2, practice forks into '*Practice by solving problems from the textbook(s)*' (Pra1) and '*Practice by developing code blocks or applications for novel situations*' (Pra2). The two types of practice correlate. Specifically, R1, Pra1, and Pra2 form a cyclic chain of study times (see Figure 1, Cluster B1). For every computer programming concept, R1, Pra1, and Pra2 possibly occur first in that order. However, subsequent series of actions do not necessarily remain directed but occur randomly based on an individual's need and perception on how to satisfy that need. Next, each practice thread independently follows its own chain of actions to achieve a learning objective. For instance, a learner engaged in Pra1 is likely to seek help from computer programmers, and this approach is associated with seeking help from programming language news groups. This thread of Cluster B makes sense because students begin by reading the course's assigned textbook. Subsequently, practice using the software development tool is initiated. Here, help is sought from professionals in the field, i.e., computer programmers, and the help of news groups is most likely sought should new questions arise. One explanation for this behavior is that professionals cannot be perpetually pounded with queries. Seeking help from people, even via social networks, highlights the importance of the social learning theory. [6] study showed that students most commonly follow the social aspect despite the diverse strategies. Engagement in Pra1 positively correlates with engagement in Pra2. Furthermore, [6] study shed light on the importance of 'inventing tasks above and beyond the assigned work'. The Pra2 subsequent chain of action is independent from that of Pra1. [34] stated that changing programs is a different goal than writing ones from scratch. Cluster B includes the highest correlation, which was between '*Learning how to use the development environment*' (Co1) and '*Learning how to use the development environment's editor*' (Co2). A possible explanation is that the more users (here, a learner enrolled in a computer programming course) become acquainted with a software (here a development tool specified in the course syllabus), the more they use its most essential workspace (here, the development tool's editor for writing code). In this case, the user introduces experience with Co1 to Co2. Cluster B visualizes a learning structure typical of learning computer programming.

The association between '*Reading the course's assigned textbooks'* and '*Practice by solving problems from the textbook(s)*' reflects the nature of learning computer programming, which is also adopted by textbooks for novices in which an explanation of a concept is exemplified by a code block. Typically, a learner reads about a concept and uses a development tool to solve problems related to it using the assigned development tool. However, the medium strength relationship might implicitly point to another approach whereby learners jump directly to practice by either copying code blocks presented by instructors or found in the textbook [37] or solving problems without doing any reading. This possibility is supported by the fact that practice was associated with '*Seeking help from computer programmers*', while the latter was not associated with reading. Noticeably, reading was only associated with practice. Nevertheless, the importance of the synchronistic approach to learning computer programming is important to highlight, which supports the analysis arrived at in section 4.2.



Note. R1: Reading the course's assigned textbooks; R3: Reading online resources; Pra1: Practice by solving problems from the textbook(s); Pra2: Practice by developing code blocks or applications for novel situations; H1: Seeking help from my instructor; H2: Seeking help from computer programmers; H3: Seeking help from programming language news groups; Cl1: Participation; Cl2: In-class understanding of computer programming concepts; Pro1: Analytical thinking; Pro2: Learning how to convert the problem solution to code; Pro3: Learning to locate and solve syntactical errors; Pro4: Learning to locate and solve logical errors; Pro5: Learning to locate and solve run-time errors; Pro6: Learning about and using objects, methods and functions; Co1: Learning how to use the development environment; Co2: Learning how to use the development environment's editor.

Figure 1. ***Study times clustered by medium and large strength associations***

This analysis is also supported by '*Practice by developing code blocks or applications for novel situations*', which associated with not only '*Reading the course's assigned textbooks*' but also '*Reading online resources'*, '*Practice by developing code blocks or applications for novel situations*', and '*Learning how to use the development environment*'. Likewise, the study by [10] presented practice, resources, review, and textbook as advice categories among others within the computer programming general study classification. In the current study, these advices are combined in a learning structure that plausibly reflects one state of a learner attempting to construct knowledge.

Cluster C is the most complex. It is a five-level mapping of study times. Its commonality with Cluster B is limited to the first two levels. At level 1, it has one core '*Learning about and using objects, methods and functions*' (Pro6), which is an essential computer programming topic. This finding agrees with [16] research, which concluded that good performance in object-oriented programming is strongly associated with a strategic learning approach. Pro6 then forks to two study times '*Learning how to convert the problem solution to code*' (Pro2) and '*Analytical thinking*' (Pro1). Pro1 and Pro2 are both associated with Pro3, Pro4, and Pro6, but not Pro5. This cluster contains several cyclical sub-clusters with three study times, such as Sub-cluster C1 and sub-cluster C2 (see Figure 1). At the macro level, a problem solver is confined by a learning structure and sociocultural factors. At the micro level, the non-directed cyclic and two-way interactions among study times of a learning structure depict how learners can traverse numerous different pathways and actions to construct knowledge. The downside of this structure's complexity accounts for [7] observation that even a bright student can get lost in writing a simple program in an introductory course. For instance, creating non-viable representations of objects at early stages in an introductory course will lead to the construction of incorrect concepts thereafter. The problem solving life cycle may continue to function as depicted in the learning structure of Custer C and its sub-clusters but with incorrect information and misconceptions. Thus, incorrect concepts should never be constructed, especially in introductory level courses [22]. Furthermore, [38] showed that syntactical issues generally hinder immediate programming productivity in Pro3. The learning structure derived from Cluster C via typical routes of learning computer programming can have variants applicable to other disciplines, such as mathematics.

4.1.4 Confirmatory Factor Analysis

A confirmatory factor analysis (*CFA*) was performed in a structural equation modeling (*SEM*) with IBM SPSS Amos (Analysis of Moment Structures) Graphics 20.0 for visual *SEM* to test the structure underlying the set of 20 study times. The endogenous variable was grade, and the 20 study times were considered the exogenous variables used to predict grade. Prior to assessing the model's fit, error covariances were included between the items with high and medium correlations, as depicted in Table 4 and visually illustrated in Figure 1. Both the chi-square to degrees of freedom ratio ((1.865), $p < .001$) [39] and RMSEA (.06) indicated an adequate fit model. However, because 1) the normed fix index (.75), the comparative fix index (.86), the Tucker-Lewis index (.81), and the goodness-of-fit index (.88) were very close to the cut-off value of .90, and 2) the standardized root mean square residual (.1073) exceeded the upper limit of .05, the model could be improved. Investigating and improving this learning model in the future would be extremely beneficial.

4.2 Results and analysis of qualitative data

Ninety-three participants chose to provide their feedback to one or more of the open-ended questionnaire items. Comments were entered into a plain text file, and one file was generated per participant. Consequently, 93 text files were created and stored in one folder. This folder was backed up. Each file name was a combination of the word 'Case' and a number, such as 'Case 1'. As a result, the number part of the name appeared on the source questionnaire and the first cell of its corresponding row in the SPSS file. Next, HyperRESEARCH was used to read and code all 93 textual source files. HyperRESEARCH is a software tool for qualitative analysis. The themes were derived from recurring patterns in the entire set of source files.

The reading open-ended question produced the highest number of responses. Respondents wrote that they relied on reading 1) PowerPoint slides distributed by their instructors, 2) notes taken in class, and 3) online resources. Although the online option was available under the reading rubric, some respondents chose to emphasize it by specifying the type of online material they read, such as articles, documents, forums, and examples.

The practice open-ended question yielded the second highest number of comments. Respondents wrote that they 1) practiced problems provided by the instructor; 2) typed the code used in class, changed it, and added code to it; 3) practiced new ideas; 4) solved assignments of other sections; 5) searched for and solved online problems; 6) practiced ideas needed at work; 7) searched for solutions to textbook exercises not solved by the instructor; 8) worked in groups with friends; 9) practiced at a learning center; 10) practiced by perfecting assignments; 11) practiced with an expert; and 12) should have done their assignments. All of the aforementioned comments emphasize the wealth of methods available for practice. Unfortunately, not all learners are aware of these methods. [5] stated that "the more practical and concrete the learning situations and materials are, the more learning takes place." Practice items 2, 3, 4, 5, 6, 7, and 10 are supported by [30] work, which adopted a strategy that encouraged students to exert additional effort by exceeding the requirements. In [2], results support practice items 1, 2, 4, and 10. [17] work supports practice items 2, 8, and 10. [34] reported the addition of practice item two as a part of a learning objective of a newly developed introductory course on media computation.

Comments added to the help open-ended question emphasized the use of online computer programming forums and libraries. The use of practice banks that give instant feedback [17] and permit students to identify areas of weakness to remedy them is equally important. Furthermore, a few students suggested solving previous exams.

The most important comments on problem solving were the transfer of knowledge to another programming language and helping other people on the internet and face-to-face. Illustrating in-class examples on the transfer of knowledge is motivating and informative to some students. For instance, illustrating a selection statement, such as 'If', written in several programming languages. Furthermore, instructors can ask capable students to help struggling students. This approach is referred to as pair programming and is encouraged by many educators as an effective learning tool [3, 20]. Comments added to the complementary learning skills open-ended question emphasized the importance of learning the logic of programming.

Comments collected from answers to the overall open-ended question were found to be best organized using an attributional approach because this approach calls for the analysis of ascriptions associated with behavior based on three dimensions: internal/external, stable/unstable, and controllable/uncontrollable [40, 41, 42]. The analysis produced five patterns. While students ascribed their problematic learning to themselves in two of these patterns, they ascribed the remaining three to external factors, including the programming course structure, instructors, and program of study. The ascribed internal factors fell under two rubrics: effort and learning strategy. Both rubrics were considered unstable and controllable, indicating that students who made the comments most probably thought that they were in control of their learning, and their achievement might be better in their future learning endeavors if they increase their efforts or follow the appropriate learning strategy. The external factors were either stable and uncontrollable, such as the program of study, or unstable and uncontrollable, including the course structure and instructor. Students who ascribed their problematic learning to their program of study most likely did not feel that they could have achieved better. Students who thought that their achievement did not meet their expectation and ascribed it to their instructors or course structure were not hopeless cases because they could have done better with another instructor or with the same instructor but a different course structure. Furthermore, the comments were tallied. As a result, each number preceding a comment in the listings below depicts how frequently the comment was mentioned relative to all other comments. This ordering started with number 1, which was associated with the most frequent comment, and ended with number 13, which was associated with the least frequent comment. The five patterns are listed below.

Internal, unstable, controllable – Effort
1) I should have devoted more time to practice solving problems
3) Negligence, I should have studied more
6) I should have studied using computers immediately after class

Internal, unstable, controllable – Learning strategy
11) My learning strategy was nil

External, unstable, uncontrollable – Course structure
2) Exams should have been carried out on a computer using the development tool
4) More in-class practice is needed
5) Live applications using the development tool should have been illustrated in class
7) Practice in computer labs is needed
13) Programmers outside the class need to provide guidance

External, unstable, uncontrollable - Instructor
9) I wish I had a more competent instructor
10) My instructor did not correct and return assignments before exams dates for us to learn from our mistakes

External, stable, uncontrollable – Program of study
8) More programming classes are needed
12) Divide the programming course according to majors

[28] work supports comment number four, i.e., “more in-class practice is needed”. They showed that hands-on practice in class is a promising instructional approach. Furthermore, [41] identified item 11, i.e., learning strategy, as the number one cause of academic achievement. These listing are crucial to the improvement of the three-cluster learning model. For instance, studying the effect of increasing in-class practice and/or introducing sessions of practice in computer labs on Cluster B would be beneficial.

5. CONCLUSION

The factual model that emerged from the data and was based on previous research is the most significant finding to advancing the body of scientific knowledge on learning computer programming. The learning model consisted of three independent structures that included 17 study times. The CFA executed in a SEM using Amos Graphics produced several indicators that supported the learning model. Nevertheless, other indicators suggested that the model could be improved. Thus, this research effort led to a learning model that serves as a good starting point for replications. However, the model is far from being complete, and considerable work remains to be done. To improve the model, this study can be replicated with different 1) faculties, departments, tracks, and computer courses; 2) sample characteristics, such age range, gender distribution, and sample size; 3) data gathering strategies; 4) methodologies; i.e. quantitative, qualitative, or both; and 5) statistical techniques, statistical packages, or data mining. Consequently, future research should investigate the learning model to improve it and thus result in the betterment of learning computer programming, which is a worldwide concern, as well as other disciplines, such as mathematics. Improvements may include the amendment of a three-cluster model structure, the study times, and the relationships.

This research effort yielded other important findings. First, while the majority of students were fully dedicated to in-class participation to understand computer programming concepts, many did not engage after class with the same enthusiasm displayed in their in-class activities. This phenomenon must be studied further because it embraces a perpetual engage/disengage studying habit that does not foster learning. Students should enhance the learning experiences they are exposed to within the walls of classrooms by studying outside the classroom on a daily basis (before and after class sessions). Adapting oneself to a daily study routine by bridging classroom sessions that consist of short learning intervals with daily study that consists of relatively longer learning intervals leads to lifelong learning. Second, respondents attached the highest importance in problem solving to ‘*Analytical thinking*’ and ‘*Learning how to convert the problem solution to code*’, in that order. Finally, the exploration of the relationships among study times revealed the importance of a synchronistic approach to learning computer programming.

The three-cluster factual model and the other findings helped to shed some light on the Lebanese educational environment in higher education. This study is unique in the Arab world, whose population is approximately 370 million. Hopefully, this attempt will serve as a stepping-stone towards building appropriate and effective computer programming learning strategies based on a factual learning model, especially for novice learners.

6. REFERENCES

[1] Boyle, R., Carter, J., & Clark, M. (2002). What makes them succeed? Entry, progression and graduation in computer science, *Journal of Further and Higher Education*, *26*(1), 3-18.

[2] Caspersen, M., & Bennedsen, J. (2007). Instructional design of a programming course: A learning theoretic approach. *Proceedings of the Third International Workshop on Computing Education Research* (pp. 111-122). Atlanta, Georgia, USA: ACM Press.

[3] Hanks, B., & Brandt, M. (2009). Successful and unsuccessful problem solving approaches of novice programmers. *Proceedings of the 40th SIGCSE technical symposium on computing science education* (pp. 24-28). Chattanooga, Tennessee, USA: ACM Press.

[4] Kinnunen, P., & Malmi, L. (2008). CS minors in a CS1 course. *Proceedings of the Fourth International Workshop on Computing Education Research* (pp. 79-90). Sydney, Australia: ACM Press.

[5] Lahtinen, E., Ala-Mutka, K., & Järvinen, H. (2005). A study of the difficulties of novice programmers. *Proceedings of the 10th annual SIGCSE conference on innovation and technology in computer science education* (pp. 14-18). Monte de Caparica, Portugal: ACM Press.

[6] McCartney, R., Eckerdal, A., Moström, J. E., Sanders, K., & Zander, C. (2007). Successful students’ strategies for getting unstuck. *Proceedings of the 12th annual SIGCSE conference on innovation and technology in computer science education* (pp. 156-160). Dundee, Scotland, UK: ACM Press.

[7] Proulx, V. (2000). Programming patterns and design patterns in the introductory computer science course. *Proceedings of the 31st SIGCSE technical symposium on*

computing science education (pp. 80-84). Austin, TX, USA: ACM Press.

[8] Thomas, L., Ratcliffe, M., Woodbury, J., & Jarman, E. (2002). Learning styles and performances in the introductory programming sequence. *Proceedings of the 33rd SIGCSE technical symposium on computing science education* (pp. 33-37). Covington, Kentucky, USA: ACM Press.

[9] Gehringer, E., & Miller, C. (2009). Student-generated active-learning exercises. *Proceedings of the 40th SIGCSE technical symposium on computing science education* (pp. 81-85). Chattanooga, Tennessee, USA: ACM Press.

[10] Hanks, B., Murphy, L., Simon, B., McCauley, R., & Zander, C. (2009). CS1 students speak: Advice for students by students. *Proceedings of the 40th SIGCSE technical symposium on computing science education* (pp. 19-23). Chattanooga, Tennessee, USA: ACM Press.

[11] Ma, L., Ferguson, J., Roper, M., & Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education*, *21*(1), 57-80.

[12] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007). *A survey of literature on the teaching of introductory programming*. ACM SIGCSE Bulletin, 39(4), 204-223.

[13] Bergin, S., & Reilly, R. (2005). Programming: Factors that influence success. *Proceedings of the 36th SIGCSE technical symposium on computing science education* (pp. 411-415). St. Louis, Missouri, USA: ACM Press.

[14] Bergin, S., Reilly, R., & Traynor, D. (2005). Examining the role of self-regulated learning on introductory programming performance. *Proceedings of the First International Computing Education Research Workshop* (pp. 81-86). Seattle, Washington, USA: ACM Press.

[15] Hawi, N.S. (2010a). Causal attributions of success and failure made by undergraduate students in an introductory level computer programming course. *Computers & Education*, *54*(4), 1127-1136.

[16] Hughes, J., & Peiris, D. (2006). ASSISTing CS1 students to learn: Learning approaches and object-oriented programming. *Proceedings of the 11th annual SIGCSE conference on innovation and technology in computer science education* (pp. 275-279). Bologna, Italy: ACM Press.

[17] Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. *Proceedings of the 9th annual SIGCSE conference on innovation and technology in computer science education* (pp. 171-175). Leeds, UK: ACM Press.

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

[19] Wiedenbeck, S. (2005). Factors affecting the success of non-majors in learning to program. *Proceedings of the First International Computing Education Research Workshop* (pp. 13-24). Seattle, Washington, USA: ACM Press.

[20] Begel, A., & Simon, B. (2008). Novice software developers, all over again. *Proceedings of the Fourth International Workshop on Computing Education Research*, (pp. 3-14). Sydney, Australia: ACM Press.

[21] Sancho-Thomas, P., Fuentes-Fernández, R., & Fernández-Manjón, B. (2009). Learning teamwork skills in university programming courses. *Computers & Education*, *53*(2), 517-531.

[22] Lui, A. K., Kwan, R., Poon, M., & Cheung, Y. H. (2004). *Saving weak programming students: applying constructivism in a first programming course*. ACM SIGCSE Bulletin, 36(2), 72–76.

[23] Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming – Views of students and tutors. *Education and Information Technologies*, *7*(1), 55-66.

[24] Schulte, C., & Bennedsen, J. (2006). What do teachers teach in introductory programming? *Proceedings of the Second International Computing Education Research Workshop* (pp. 17-28), Canterbury, UK.

[25] Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. *Proceedings of the 6th annual SIGCSE conference on innovation and technology in computer science education* (pp. 49-53). Canterbury, UK: ACM Press.

[26] Rountree, N., Rountree, J., & Robins, A. (2002). *Predictors of success and failure in a CS1 course*. ACM SIGCSE Bulletin, 34(4), 121-4.

[27] Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: A study of twelve factors. *Proceedings of the 32rd SIGCSE technical symposium on computing science education* (pp. 184-188). Charlotte, NC, USA: ACM Press.

[28] Boyer, K. E., Phillips, R., Wallis, M. D., Vouk, M. A., & Lester, J. C. (2009). The impact of instructor initiative on student learning: A tutoring study. *Proceedings of the 40th SIGCSE technical symposium on computing science education* (pp. 14-18). Chattanooga, Tennessee, USA: ACM Press.

[29] Hawi, N. (2010b). The exploration of student-centered approaches for the improvement of learning programming in higher education. *US-China Education Review*, 7(9), 47-57.

[30] Roberts, E. (2000). Strategies for Encouraging Individual Achievement in Introductory Computer Science Courses. *Proceedings of the 31st SIGCSE technical symposium on computing science education* (pp. 295-299). Austin, TX, USA: ACM Press.

[31] Xiaohui, H. (2006). Improving teaching in computer programming by adopting student-centred learning strategies. *The China Papers*, 6, 46-51.

[32] Tavakol, M., & Dennick, R. (2011). Making sense of Cronbach's alpha. *International journal of medical education*, 2, 53-55.

[33] Seitz, Lindsey (2012). Student attitudes toward reading: A case study. *Journal of Inquiry and Action in Education*, *3*(2), 30-44.

[34] Guzdial, M. (2003). *A media computation course for non-majors*. ACM SIGCSE Bulletin, 35(3), 104–108.

[35] Betts, G. T. (1985). *Autonomous learner model for the gifted and talented*. Greeley, CO: ALPS Publishing.

[36] Gasparinatou, A. & Grigoriadou, M. (2011). Supporting students' learning in the domain of computer science. *Computing Science Education*, *21*(1), 1-28.

[37] Kaasbøll, J., Berge, O., Borge, R. E., Fjuk, A., Holmboe, C., & Samuelsen, T. (2004). Learning Object-Oriented Programming. *Proceedings of the 16th Workshop of the Psychology of Programming Interest Group* (86-96). Carlow, Ireland. Retrieved from http://www.ppig.org/ papers/16th-kaasboll.pdf

[38] Bennedsen, J. & Caspersen, M. (2012). Persistence of elementary programming skills. *Computer Science Education*, *22*(2), 81-107.

[39] Carmines, E. G., & McIver, J. P. (1981). Analyzing models with unobserved variables. In G. W. Bohrnstedt & E. F. Borgatta (Eds.), *Social measurement: Current issues*. Beverly Hills: Sage.

[40] Griffin, K. A. (2006). Striving for success: A qualitative exploration of competing theories of high-achieving black college students' academic motivation. *Journal of College Student Development, 47*(4), 384-400.

[41] Hawi, N. S. (2008). An *attributional approach to computer programming achievement of undergraduate business computing students in a university computer science department* (Doctoral dissertation). Retrieved from British Library ETHOS. (Accession http://hdl.handle.net/2381/8219).

[42] Weiner, B. (2000). Intrapersonal and interpersonal theories of motivation from an attributional perspective. *Educational Psychology Review, 12*, 1-4.