Cognitive Characteristics that Influence Effectiveness of Students during Computer Programming Process

V.G. Renumol, D. Janakiram and S. Jayaprakash
Department of Computer Science and Engineering,
Indian Institute of Technology Madras,
Chennai, India.

Abstract

What are the cognitive characteristics that influence the effectiveness of a student during the computer programming process? To investigate this question, a comparative study has been conducted on verbal protocols collected from a set of effective and ineffective students. It is observed that there are differences in certain cognitive characteristics, among the groups. Also there are variations in the effectiveness and sequence of subtasks during the process. A well-defined pattern is observed with the effective students. The study has concluded that certain procedural cognitive concepts are critical to decide the effectiveness during programming; in addition to the knowledge of programming language constructs.

IJERT

1. Introduction

Computer programming is a highly rewarding skill. But, teaching as well as learning of computer programming is a challenge in Computing Education [McGettrick et al. 2005]. Different publications in programming education show that the failure rate and drop-out rate of programming courses are relatively high [Robins *et al.* 2003; Mancy and Reid 2004; McGettrick *et al.* 2005] and the overall effectiveness is poor. Even though there are motivational factors to learn programming [Jenkins 2001], many novices find it very difficult to learn programming. Various studies have been conducted by researchers on different aspects of programming education to increase the quality of programming education, but even after forty years of research, there are gaps in programming education. Even though programming is mainly a cognitive task, studies on programming education from cognitive perspective are very rare. Hence the main objective of this study is to investigate on cognitive characteristics of students during programming process.

The students in programming course can be broadly categorized as effective and ineffective based on their effectiveness in programming. Effective students can write programs and they typically learn programming with moderate effort. Most of the time, programming is a self-discovered process for them. Whereas, the ineffective students cannot write correct programs and need more personal attention and cognitive support to learn programming. Since the failure rate is high in programming courses, the ineffective category plays a significant

role in the effectiveness of programming courses. This study tried to find out the cognitive behavioral characteristics that differentiate these two groups during the programming process. The study has been conducted only to characterize the state of mind of the participants with respect to effectiveness during the programming process and not to investigate on how the participants become effective or ineffective over a period of time, which is beyond the scope of this study.

The article is organized as follows: Section 2 describes the related work. Section 3 describes the methodology - verbal protocol collection and classification. Section 4 describes the qualitative analysis and section 5 describes the quantitative analysis. Finally section 6 concludes the article.

2. Related Work

From the literature survey it is obvious that many studies have been conducted on various aspects of programming education to improve its effectiveness. For example, studies on development of teaching strategies for programming courses [Winslow 1996; Campbell and Bolker 2002; Robins *et al.* 2003; Bennedsen and Caspersen 2005; Caspersen and Bennedsen 2007; Alkhalifa 2008; Caspersen and Kolling 2009], difficulties in learning various features of a programming language [Milne and Rowe 2002; Lahtinen *et al.* 2005; Garner, Haden and Robins 2005], development of interactive multimedia tools to help students learn programming [Al-Imamy, Alizadeh and Nour 2006; El-Zein, Langrish and Balaam 2007; Lee, Pradhan and Dalgarno 2008], failure rates in Introductory Programming Courses (IPC) [Bennedsen and Caspersen 2007], predicting programming skill of students [Mazlack 1980; Mancy and Reid 2004; de Raadt *et al.* 2005; Dehnadi and Bornat 2006; Simon *et al.* 2006], programming skill evaluation [Farrow and King 2008], first programming language to teach [Mclver and Conway 1996; Robins, Haden and Garner 2006; Kaplan 2010], plagiarism in programming [Joy and Luck 1999; Bowyer and Hall 2001; Goel and Rao 2008; Rosales *et al.* 2008] etc.

Programming is a human-centered activity where one needs to design a solution for a given problem and then translate it into a sequence of computer language instructions. But, most of the programming languages are designed from machine point-of-view, considering machine efficiency related issues rather than human-related issues [Shneiderman 1975]. Hence, programming demands different cognitive skills [Gael 2005; Xu and Rajlich 2004], which are generally not taught in programming courses. Typically, teachers spend most of the time to teach the programming language syntax and semantics [Caspersen and Kolling, 2009]. They show readymade programs in the class and explain the execution sequence or algorithm, where they are necessitated to show the program-development process. Moreover, students use the text books, which generally show correct and complete programs with intermediate outputs. But being a static media they are not able to show the dynamic program-development process [Bennedsen and Caspersen, 2005]. Students can use it mainly to acquire factual knowledge. However, the

IJERT

learning curve for different programming languages varies based on the language and the cognitive characteristics of the learner [White and Sivitanides 2002]. And most of the programming languages are developed from professional point of view rather than an educational point of view. All the above factors increase the difficulty for the novices to learn programming, which in turn increases the challenge to programming educators.

Looking this challenge from a cognitive perspective may improve the situation, because programming necessitates more cognitive effort from the learner to become an effective programmer. However, studies from the cognitive perspective are very rare in programming education. A previous study by the authors [Renumol et al. 2010] was an exploration to identify the cognitive processes (CPs) of the effective and the ineffective students during the programming process (using C language). CP is a process in the brain, which does information retrieval and/or information storage and/or information processing. The above-mentioned study identified a set of 42 CPs relevant for both the effective and the ineffective students. Basically, this result gives knowledge on the CPs of the programming process and can make teachers aware of the cognitive difficulty (due to many CPs) to learn programming. But, if the effective and the ineffective students have the same set of CPs, then what makes them different during the programming process? This study is an attempt to answer it using the methodology Verbal Protocols Analysis (VPA), which is explained in the next section.

3. Verbal Protocol Collection And Classification

Verbal Protocols (VPs) are oral records of the thought process of a participant during a task. It can be recorded while performing the task (concurrent verbal protocol) or immediately after the task (retrospective verbal protocol) [Ericsson and Simon 1984/1993]. Concurrent verbalization is better over retrospective for long duration tasks [Kuusela and Paul 2000] like programming. Therefore, in this study the participants were asked to verbalize their thoughts during the programming task and the experimenter recorded it as audio files. The VP files are then used with other artifacts for the analysis.

VP Collection: Generally, a small sample size is feasible for think aloud methods due to high cost and time for the protocol analysis [Davis and Bistodeau 1993; Hungerford et al. 2004; Isenberg 1986; Krahmer and Ummelen 2004]. However, in this study 45 VPs have been collected in order to improve the quality of analysis, especially the quantitative analysis. Moreover, it can provide a rich data for the qualitative analysis. The participants were undergraduates, postgraduates or research scholars in computer science and engineering and have undergone a programming course in C language. Their experience in 'C' language ranges from one to many years, which indicates the number of years since they started learning 'C' programming language. That is, some traces have been there in their brain since then. However, studies do not support a strong relation between experience and effectiveness [Hungerford et al. 2004]. This cognitive study gives importance to their

IJERT

effectiveness rather than experience or demographic details, because it attempts to characterize the strategy and state of the mind with respect to effectiveness during the task.

The experiment was conducted at an individual level with the paper-and-pencil method. There was no time constraint to complete the task. The environment for the experiment was noise-free, realistic, and comfortable. The experimenter instructed the participants about the experiment beforehand and clarified their doubts related to the experiment. There were 30 programming questions taken from a regarded text book of C programming [Gottfried 1991] and are given in the appendix A of [Renumol et al. 2010]. The experimenter asked the participants to select a question of his/her choice which has not been memorized by the subject. A program which is already memorized by the subject will not reveal the actual effectiveness of the subject. It will be a recollection process only. Once they selected a question, immediately they started solving it and verbalized their thoughts during the entire programming process and the experimenter recorded it as an audio file.

The study used concurrent verbalization and it often lacks completeness since it is done along with the task because the subjects tend to be silent when they concentrate more on the primary task of programming. This leads to an incomplete and incoherent VP. Hence the experimenter kept on reminding the participants to keep talking when the silent pauses exceeded 10-15 seconds and also avoided social communication.

VP Classification: After the collection of the data, there are mainly two artifacts - VP as audio files and the participants' worksheets. The latter will contain the rough work, the intermediate steps and the final program of the participants. To classify the participants as effective or ineffective, only the final programs are needed. Each of the programs written on worksheets has been edited and compiled to check the effectiveness of the participant. Programs with logical and syntax errors were put in the ineffective category. Some of them were incomplete, showing their ineffectiveness. The programs which compiled properly have been executed to see the correctness of the logic. Those run correctly were put in the effective category. During this process, the experimenter corrected some trivial syntax mistakes such as missing semicolon, brackets etc. which the participant also would have done, provided a computer to do the task. Finally, 25 students have been classified as effective and 20 as ineffective out of the 45 students participated in the experiment.

After the collection and classification of the VPs, two types of analyses have been conducted on the data - qualitative and quantitative. They are discussed in the following sections.

4. Qualitative Analysis And Observations

It has been decided to employ content analysis as the methodology for the qualitative analysis. The analysis and its results are explained in the following section.

IJERT

4.1 Content analysis

It is a methodology to study the content of communication and is widely used in Social Science research [Krippendorff 2004]. The analyst systematically goes through the communication artifact to find out the properties of the content by analyzing who says what, why, in which context, with what mental state, whether certain words are repeated, the diction, the action, the punctuation etc. The communication artifact can be textual, audio or video information. In this study the analyst has done content analysis on the transcriptions and/or the audio VP files of both the groups by going through them several times to infer some patterns and trends. The worksheet also has been referred whenever needed, for example, to get the context, the action etc. The analyst has observed that there are intensity variations in CPs, difference in the usage of higher and lower CPs, and variations in the effectiveness and sequence of subtasks, among the groups. These observations are described in the following subsections.

4.1.1 Intensity Variation of CPs. It is observed that there are variations in intensity of the CPs, which means that there are variations in the potency of the CPs, among the groups. For example, abstraction, translation, induction, deduction, comprehension, analysis, synthesis etc. were stronger and faster in the effective students and weaker and slower in the ineffective. Thus these CPs seem to support programming process. On the other hand, the CP, confusion was stronger with the latter than the former. The analyst could find several such instances of different CPs from the collected VPs. Some example excerpts for intensity variations in CPs like abstraction, translation and confusion are given in Table I.

IJERT

Table I. Examples for intensity variations

Cognitive Process	Excerpt from effective subject	Excerpt from ineffective subject
Abstraction	So first I will draw a sample output to see how it looks, the figure A….(subject is drawing). So I find that the height 'h' is equal to the initial number of blank spaces. It is denoted by 'x'. So what I should have is; for each line there should be one less number of blank spaces, in the beginning. And considering the number of characters, it should begin with one character in the first line and for the further lines; it should be incremented by 2. so …now the logic is clear.. [subject is able to	So we have to get this thing: One * and then one * on left side and one on right side. One again, one in the middle, no, one in……(subject draws) this kind of thing ……..…so what do I need? ...one row will increase like this ….. one row will increase like this and the other will like this and the starting element, it is like array; not array exactly. Like… 1,2,3,4,5,6; leaving this space blank, this space middle blank……..hm…I need more……so first of all I need 2 variables: one varying like this and one varying like this.(subject showing x axis and y axis) [subject is not able to abstract correctly]

	abstract things easily]	
Translation	So if head == null create memory for that. So head=; we have to use the malloc() function. (struct node *) malloc(size(struct node)). Current is made as current's next node. Current=current.next. [subject is able to translate]	We have allocated the space for the elements, then will write the loop like: for(x=1; x<=n; x++) and y from 1 to n; y++, ah…so…if… if x=y then …… .so basically I am thinking over what exactly is the code I should write. [subject is not able to translate]
Confusion	I can write a loop here itself ……..so for(j= it always starts with s; j<……..;++j, in this case, 0, 4-?, s position is? [Confusion] Ok. i=0, n=4, 4-0=4; j<4. it is …(murmering) starts with s+1; this is s+1, position after this; j< that is, 4-? i in this case is 1, that is 3…… n is 4, 4-i; i is 0; j<4. ha. this is fine. Ok. So now have the…..sorry..this is c. so now we have the 'for' loop to iterate through column values [This excerpt shows that though the subject had slight confusion, but able to overcome it and proceed with the task]	….start for loop again i=0;i<j; equal should come or not? i=1 ; i<j; then i should increase…….i should always print in the, which position? 1,2 ….it should always print in the what? n, this is what? n,..n,1…..I can't do that also……..(murmuring)…….n-1 spaces ……..no I am not getting……I am not getting what should I do….how should I use the ….main funda I am not getting. [This excerpt shows that the subject is undergoing confusion and not able to proceed with the task. Confusion is very intense here]

4.1.2 Usage of lower and higher level CPs. There are two types of CPs – lower and higher level [Wang et al. 2006]. Lower-level CPs need less cognitive effort, whereas higher CPs are conscious CPs, which put more cognitive load on the brain. Higher-level CPs are carried out with the help of one or more lower and/or higher-level CPs. Through the content analysis, the analyst could observe that the effective subjects properly use the higher and lower level CPs in order to achieve the goal and they are stronger in them. This was not the case with the ineffective subjects. For example, problem solving is a higher level CP which needs other CPs like comprehension, abstraction, analogy, decision making, reasoning, attention, analysis, synthesis etc. On the other hand, sensational CPs (vision, audition etc.), memorization,

recollection etc. are examples of lower level CPs. The ineffective subjects prefer to stay in lower-level CPs such as memorization and recollection and are weaker in higher level CPs. For instance, the ineffective students were mostly unable to explain the cause-effect relation, which is a higher CP called reasoning. Table II shows an example for this. In this example, the problem given for both the cases was matrix multiplication. One can see that the effective subject is going successfully through various CPs and the ineffective subject is unable to do it.

Table II. Usage of higher and lower-level CPs

Excerpt from an effective subject	Excerpt from an ineffective subject
So the bigger loop, outer loop should be for(i=0; i<m;i++), that is I am going over the rows of m1, then there is some second loop, which should go over….. These two outer loops should actually correspond to the final dimensions of the matrix. Right? That is m x p. The second loop should go over p. for (int j=0; j<p;j++)…….[Synthesis, Reasoning, Explanation, Comprehension] How do I do the addition part? [Interrogation] So each row, multiply each row by each column basically. I know that this is some element-wise product. [Recollection] So pick up the first element of the first row of m1, multiply it with the first element of the first column of m2 , then the second element of the first row multiplied with the second element of the column, so on…right? [Explanation] So let me start, 'i' ; ' i ' is going over the row; so for the same value of 'i' change;…….. change the value of ……..the column…. And here in this case for the same value of p change the value of the row, this is how it works. Ok? [Induction, Synthesis] This m and n are going to be the element of m3. I have to initialize ……m3[i,j] to zero here, [Synthesis] so that when I add up all the things I can just treat it as sum=sum+n….[Explanation] then I have to state one loop to 'k' such that the k value has to go over n here, [Synthesis] because that is the only dimension which is remaining and those are the number of columns of m1 and number of rows of m2. ok? [Explanation, reasoning] For each such thing, I just have to multiply it,…i j; m3[i][j]=m3[i][j]+m1[i][k]*m2[k][j]. … [Synthesis, Translation]	For this recursion for(i=0;i<=3..so for ar[0] we are having r1 and for ar[2] we are having r2 and so on.. Next we are taking 3 elements and go on multiplication till this loop is continued. So the multiplication is done like multiplication condition, mult1=(r1*c1) and (r2*c2) and (r3*c3), mult2=(r2*c1), (r2*c2),(r3*c3); mult3=(r3*c1), (r3*c2),(r3*c3). Next after scanf("%d",…we just print r1, r2 and r3. so the printf will give the result of mult1, mult2 and mult3 [The subject does not have any reasoning for the statements, seems he is trying to recollect matrix multiplication rather than synthesizing. It seems that the subject has forgotten the language syntax and not able to translate what he knows about matrix multiplication]

IJERT

4.1.3 Subtasks and their effectiveness. By the content analysis, the analyst could also observe various subtasks the subjects do during the programming process. They are, in general, problem understanding, design, coding, testing and iteration. In problem understanding phase the participants try to select and comprehend a question from the given list of questions. This takes one or two minutes. In the design phase they try to plan the solution, in the coding phase they translate the design to a program and in the testing phase they check whether the program is correct or not. If it is not correct they have to modify the design or code accordingly. For this, they iterate through the design or code and modify it. These last four phases take few minutes to hours to complete, based on how effective the subject is.

All the 45 VPs have been analyzed for the subtasks and their effectiveness. The starting and ending of each of the subtasks have been noticed from the audio files and the worksheets. Clarity of the boundary of the subtasks were more in the case of the effective subjects than that of the ineffective subjects. The latter had a tendency to mix up the subtasks, especially design and coding. However, an attempt has been made to mark the boundary of the subtasks of the ineffective subjects also, based on the context. It was observed that most of the effective subjects went through all the phases whereas most of the ineffective subjects did not go precisely through all the phases.

The analyst has evaluated the effectiveness of each of the subtasks for each of the participants and given marks out of five. This is done by hearing the audio files several times and evaluating correctness of the subtasks from the worksheets. A mark of one indicates extremely ineffective in the subtask and five indicates fully effective in the subtask. For most of the ineffective subjects what the subject called as coding was an amalgamation of coding with design. So the analyst evaluated the hidden design in the code and put marks accordingly. Such merging was very rare with the effective subjects. Marks for iteration have been put based on how many times they iterated through the process and how much useful each iteration was. Two tables have been formed from the evaluation results – one for the effective subjects with 25 rows and the other for the ineffective subjects with 20 rows. Each has columns such as design, coding, testing and iteration. The problem-understanding phase was left out from the table since nobody found any difficulty in understanding the selected problem. Two graphs have been drawn based on the values from these tables; with the subtasks on the x-axis and marks on the y-axis. The graph of the effective category is given in Fig. 1 and that of the ineffective category is given in Fig. 2.

IJERT

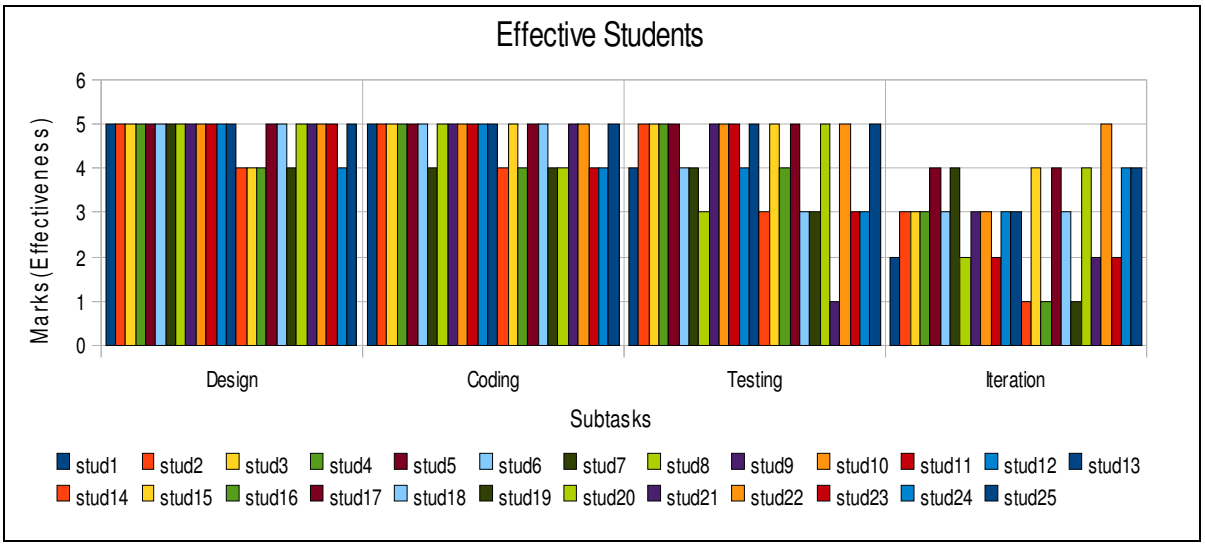

Fig. 1: Subtasks and their effectiveness of the Effective students

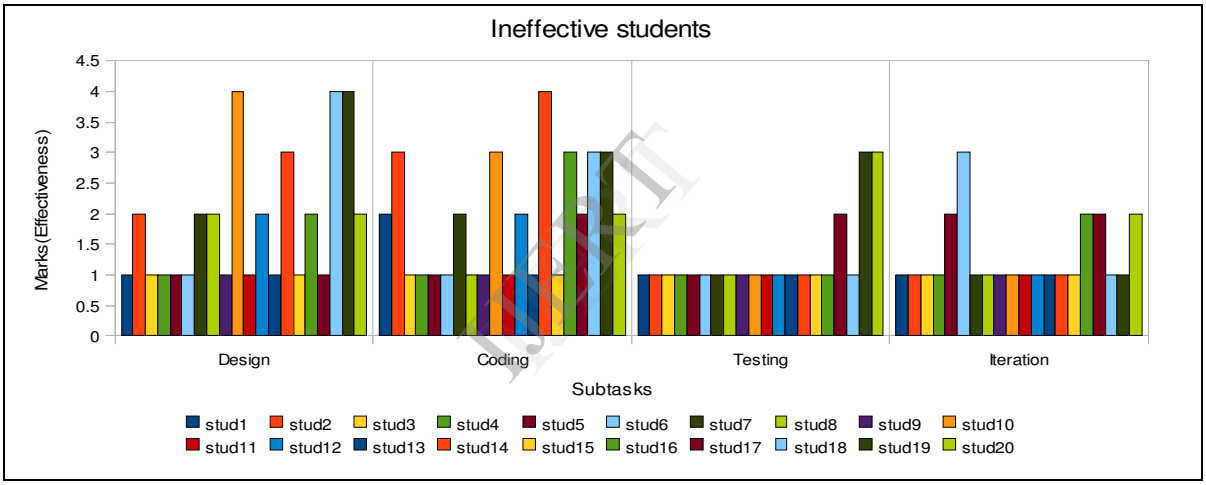

Fig. 2: Subtasks and their effectiveness of the Ineffective students

It is observed from the graphs that the effective subjects are highly effective in each of the subtasks whereas the ineffective subjects are less effective in each of the subtasks. The latter has a tendency to put more effort in coding and overlooking the other inevitable phases of program development, such as design and testing. The effective subjects iterated more times than the ineffective subjects. When the former iterated through the phases, they improved over the solution which was rare with the latter category.

4.1.4 Subtasks and their sequence. As an extension of the previous observations, analysis has been done to see the pattern of sequencing through the subtasks for both the categories. It is observed that the sequence of subtasks during programming is different for both the effective and the ineffective groups. The effective students have a sequence in the following order during the process:

1. Problem comprehension
2. Design
3. Coding
4. Testing

5. Iteration

The above pattern is observed invariably with the effective subjects whereas it is rarely observed with the ineffective subjects. The ineffective category does not have a proper sequence during the task. After reading and roughly comprehending the question, most of the ineffective subjects have hopped directly to the coding phase by skipping the design. They have a tendency to mix coding with design. They rarely have gone through design, coding, testing and iteration effectively. Fig. 3 and 4 show the pattern of the subtask sequence of the effective and the ineffective subjects respectively. In Fig.4 the thick arrows show the frequent path of an ineffective subject. That is jumping to coding phase from problem comprehension and then leading to an incorrect program. Dotted-curved arrows are the rarely traversed paths by the ineffective students.

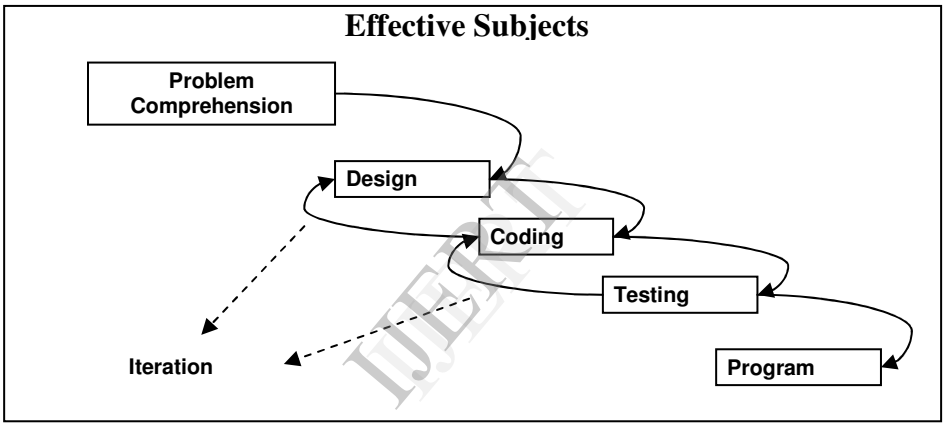

Fig. 3: Pattern of subtask sequencing by the effective category

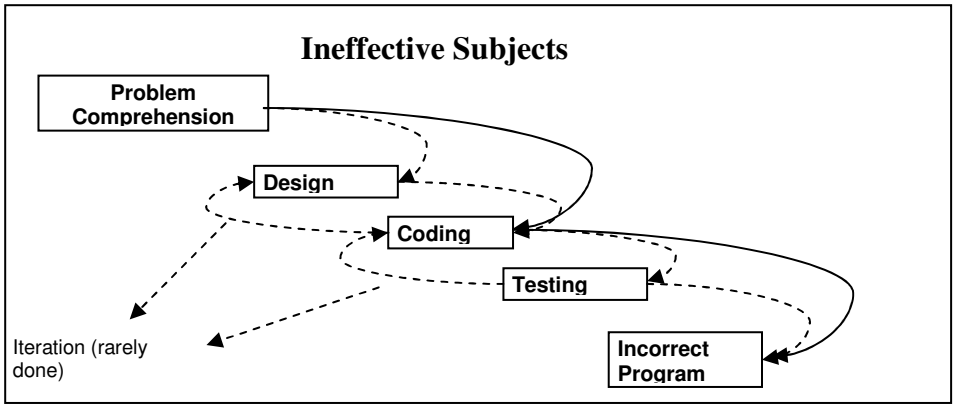

Fig. 4: Pattern of subtask sequencing by the ineffective category

Hence, it is observed that the effective subjects go through layers of abstraction (problem understanding, design, coding, testing and iteration) and they are highly effective in each layer of abstraction. Layers of abstraction reduce the complexity of the task since only a small portion of the task is handled at a time and thus the cognitive load on the brain is also reduced. Literature shows that abstraction is a key skill required for computing [Kramer 2007]. Wing [Wing 2006] also says that computational thinking requires multiple levels of abstraction. Hence, their strategy can be considered as a relevant factor for their effectiveness during the entire programming process.

On the other hand, the ineffective subjects rarely start with design after problem comprehension. But design is an important phase which links the problem domain with the solution domain. They spend most of the time in coding phase and have a tendency to mix coding with design. Also they very rarely test and debug the code. Therefore the layer of abstraction is weak during the task, which increases the complexity of the task and cognitive load on the brain. Hence they become ineffective in each of the subtasks and consecutively ineffective in the entire task.

Therefore, it was observed from the qualitative analysis of the 45 VPs that the effective students have better strategy to solve a given programming problem. They successfully go through different phases of program construction like understanding the problem, design the solution, coding and testing, in order to develop the program incrementally. Also they iterate through any of these phases to improve or correct the program. In order to corroborate this observation, a quantitative analysis has been done further, which is explained in the next section.

5. Quantitative Analysis

To substantiate the above-mentioned observations, a hypothesis was framed from the observation as follows and verified statistically.

Hypothesis: Effective students successfully go through layers of abstraction (LOA) such as problem understanding, design, coding and testing to develop the program/solution incrementally and iterate through the LOA to correct or improve the solution. Such pattern was not observed with the ineffective students.

Null Hypothesis: There is no difference between the effective and the ineffective students.

In order to test the hypothesis it was decided to conduct a quantitative analysis called discriminant analysis. It is a statistical method for examining differences between two or more groups of objects with respect to several variables simultaneously [Klecka 1980]. Discriminant models are estimated from these variables and the objects are classified using this model. In this study, effectiveness has been taken as the dependent (categorical) variable in non-metric form (1- effective, 0- ineffective) and the design, coding, testing, LOA, and iteration are taken as the independent (predictor) variables in metric form. The effectiveness of participants has already been measured by testing the final program. If the

IJERT

program executed correctly then the candidate is effective otherwise he/she is ineffective. As explained in the previous section, the analyst has evaluated the effectiveness of each of the participants in each of the subtasks (design, coding and testing) and given marks out of five. This was done by hearing the audio files several times, and by evaluating the subtasks from the worksheets. A mark of one indicates very less effective in the subtask and five indicates fully effective in the subtask. The LOA values are not available in the tables from the qualitative analysis phase. Hence it has been calculated row-wise by taking the average of the design, coding and testing marks. A value of three or more indicates that the subject has a better LOA. Thus all the independent variables are measured in a five-point Likert scale. The independent variables from the 45 VPs have been codified in metric form and made into a table of 45 rows. A sample of it (first 10 rows) is shown in Table III. The complete table has been input to the statistical software, Statistical Package for Social Sciences (SPSS) version 15.0 for discriminant analysis. The software estimated the discriminant model from the variables and classified the students as effective and ineffective, using this model.

Table III: Sample of Coded Data for Discriminant Analysis

Sl.No.	Effective	Design	Coding	Testing	LOA	Iteration
1	1	5	5	4	4.666667	2
2	1	5	5	5	5	3
3	1	5	5	5	5	3
4	1	5	5	5	5	3
5	1	5	5	5	5	4
6	1	5	5	4	4.666667	3
7	0	1	2	1	1.333333	1
8	1	5	4	4	4.333333	4
9	0	2	3	1	2	1
10	1	5	5	3	4.333333	2

IJERT

Table IV. Classification Results (a)

		Effective	Predicted Group Membership		Total
			.00	1.00	.00
Original	Count	.00	19	1	20
		1.00	0	25	25
	%	.00	95.0	5.0	100.0
		1.00	.0	100.0	100.0

(a) 97.8% of original grouped cases correctly classified.

The classification result of the software is given in Table IV. All cases from the effective group are classified correctly (100% correct classification) and one case from the ineffective group is moved (95% correct classification) to the effective group. Thus the software classified 26 students as effective and 19 students as ineffective out of the 45 cases, showing 97.8% of correctness in the classification.

In summary, the SPSS results corroborated the observation that there is difference between the effective and the ineffective programmers in their strategy. The selected predictors (design, coding, testing, LOA and iteration) are significant in differentiating the effective and the ineffective students. The effective students have better abstraction layers, they iterate through the layers to improve or correct the solution and, they are effective through each of the subtasks. On the other hand, the ineffective students did not exhibit any well-defined strategy during the task.

6. Conclusion

The study has analyzed 45 VPs to find out the disparity in the effectiveness among the effective and the ineffective students, during C programming. Content analysis was done first and observed that there are intensity variations in CPs, difference in usage of higher and lower CPs, and variations in the effectiveness and sequence of subtasks of the programming process, among the groups.

The above observations from the qualitative analysis have been statistically verified by discriminant analysis using the SPSS tool. The SPSS results corroborated the observation that there is difference between the effective and the ineffective programmers in their procedural strategy and showed 97.8% of accuracy in the classification. Therefore one can conclude that the procedural knowledge predominantly decides the effectiveness in programming task. This warrants a further study to develop a teaching strategy mainly for ineffective students, giving importance to procedural concepts. Further research with more diverse data in different programming paradigms and languages can strengthen the results of this study.

References

1. AL-IMAMY, S., ALIZADEH, J. & NOUR, M. A. 2006. On the development of a programming teaching tool: The effect of teaching by templates on the learning process. *Journal of Information Technology Education. 5*, 271–283.
2. ALKHALIFA, E. M. 2008. Sequential programming instruction and gender differences. *IEEE Transactions on Education. 51*, 4, 417–422.
3. BENNEDSEN, J. & CASPERSEN, M. E. 2005. Revealing the programming process. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'05)*. 186–190.
4. BENNEDSEN, J. & CASPERSEN, M. E. 2007. Failure rates in introductory programming. *Inroads-ACM SIGCSE Bulletin, 39*, 2, 32–36.

IJERT

5. Bowyer, K. W. & L.O. Hall (2001) Reducing Effects of Plagiarism in Programming Classes. *Journal of Information Systems Education,* 12(3), 141-148.
6. CAMPBELL, W. & BOLKER, E. 2002. Teaching programming by immersion, reading, and writing. In *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference (ASEE'02)*. 23–28.
7. CASPERSEN, M. E. & BENNEDSEN, J. 2007. Instructional design of a programming course – A learning theoretic approach. In *Proceedings of the Workshop on International Computing Education Research (ICER'07)*. 111–122.
8. CASPERSEN, M.E. & M. KOLLING (2009) STREAM: A First Programming Process. *ACM Transactions on Computing Education,* 9(1), 1-29.
9. DAVIS J. N. & BISTODEAU, L. 1993. How do L1 and L2 reading differ? Evidence from think aloud protocols. *Modern Language Journal 77*, 4, 459–472.
10. DE RAADT, M., HAMILTON, M., LISTER, R., TUTTY, J., BAKER, B., BOX, I., CUTTS, Q., FINCHER, S., HAMER, J., HADEN, P., PETRE, M., ROBINS, A., & SIMON. 2005. Approaches to learning in computer programming: Students and their effect on success. In *Proceedings of the Annual International Conference of the Higher Education Research and Development Society of Australasia (HERDSA'05)*. 407–414.
11. DEHNADI, S. & BORNAT, R. 2006. The camel has two humps. Technical report. School of Computing, Middlesex University, UK.
12. EL-ZEIN, A., LANGRISH, T., & BALAAM, N. 2007. A self-practice online tool for teaching and learning computational skills in engineering curricula. In *Proceedings of the International Conference on Engineering Education (ICEE'07)*.
13. ERICSSON, K. A. & SIMON, H. A. 1984/1993. *Protocol analysis: Verbal reports as data.* Rev. ed. Cambridge: MIT Press.
14. FARROW M. & KING, P. J. B. 2008. Experiences with online programming examinations. *IEEE Trans. Educ. 51*, 2, 251–255.
15. Gael, Y. (2005) A theory of Program Comprehension: Joining Vision Science and Program comprehension, Technical Report, GEODES, University of Montreal, Canada, December, 1-26.
16. GARNER, S., HADEN, P. & ROBINS, A. 2005. "My Program is correct But it Doesn't Run: A Preliminary Investigation of Novice Programmers' Problems", In *Proceedings of the Australasian Computing Education Conference (ACE '05)*, pp. 173-180.
17. Goel, S. & Rao, D. (2008) Plagiarism and its Detection in Programming Languages, Technical Report, Department of

IJERT

Computer Science and Information Technology, JIITU, May, 1-8.

18. GOTTFRIED, B. S. 1991. *Theory and problems of programming with C. Schaum's outline series*. Tata McGraw-Hill Edition.
19. Hungerford, B., A.R. Hevner, and R.W. Collins. 2004. *Reviewing software diagrams*: A cognitive study. IEEE Transactions on *Software* Engineering 30 (2): 82-96.
20. ISENBERG, D. J. 1986. Thinking and managing: A verbal protocol analysis of managerial problem solving. *Acad. Manage. J. 29*, 4, 775–788.
21. JENKINS, T. 2001. The Motivation of Students of Programming. *SIGCSE Bulletin,* 33(3), 53-56.
22. Joy, M. & M. Luck (1999) Plagiarism in Programming Assignments, *IEEE Transactions On Education,* 42(2), 129-133.
23. KAPLAN, R.M. 2010. Choosing a first programming language, In *Proceedings of the ACM Conference on Information Technology Education*.
24. KLECKA, W. R. 1980. *Discriminant analysis for Social Sciences*. Sage Publications, 71 pages.
25. KRAHMER, E. & UMMELEN, N. 2004. Thinking about thinking aloud: A comparison of two verbal protocols for usability testing. *IEEE Trans. Prof. Commun. 47*, 2, 105–117.
26. KRAMER, J. 2007. Is Abstraction the Key to Computing? *Communications of the ACM*. Vol. 50, No.4. 37-42.
27. KRIPPENDORFF, K. 2004. *Content analysis*: *an introduction to its methodology*, 2nd ed. Sage Publications, 413 pages.
28. KUUSELA, H. & PAUL, P. 2000. A comparison of concurrent and retrospective verbal protocol analysis. *Am. J. Psych. 113*, 3, 387–404.
29. LAHTINEN, E. ALA-MUTKA, K. & JÄRVINEN, H. 2005. A study of the difficulties of novice programmers. In *Proceedings of the 10th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'05)*. 14–18.
30. Lee, M. J. W., S. Pradhan & B. Dalgarno (2008) The Effectiveness of Screencasts and Cognitive Tools as Scaffolding for Novice Object-Oriented Programmers. *Journal of Information Technology Education,* 7, 61-80
31. Mclver, L. & D. Conway (1996) Seven Deadly Sins of Introductory Programming Language Design. Proceedings of *International Conference on Software Engineering: Education and Practice,* January, New Zealand, 1-8.
32. MANCY, R. & REID, N. 2004. Aspects of cognitive style and programming. In *Proceedings of the16th Workshop of the Psychology of Programming Interest Group (PPIG'04)*. 1–9..
33. MCGETTRICK, A., BOYLE, R., IBBET, R., LLOYD, J., LOVEGROVE, G., & MANDER, K. 2005. Grand

IJERT

challenges in computing: Education – A summary. *Computer Journal. 48*, 1, 42–48.
34. MILNE, I. & ROWE, G. 2002. Difficulties in learning and teaching programming – Views of students and tutors. *J. Educ. Inf. Technol. 7*, 1, 55–66.
35. RENUMOL, V. G., JAYAPRAKASH, S., & JANAKIRAM, D. 2009. Classification of cognitive difficulties of students to learn computer programming. Technical report No. IITM-CSE-DOS-2009-01. Distributed and Object Systems Lab, Department of Computer Science and Engineering, IIT Madras, India.
36. RENUMOL, V. G., JANAKIRAM, D., & JAYAPRAKASH, S. 2010. Identification of cognitive processes of effective and ineffective students during computer programming. *ACM Trans. Computing Education 10*, 3, Article 10, 1-21.
37. ROBINS, A., ROUNTREE, J., & ROUNTREE, N. 2003. Learning and teaching programming: A review and discussion. *Computer Sci. Educ. J. 13*, 137–172.
38. ROBINS, A., HADEN, P. & GARNER, S. 2006. Problem Distributions in a CS1 Course, In *Proceedings of the 8th Australasian Computing Education Conference (ACE'06)*.
39. Rosales, F., A. Garcia, S. Rodriguez, J. Pedraza, R. Mendez & M. M. Nieto (2008) Detection of Plagiarism in Programming Assignments , *IEEE Transactions On Education*, 51(2), 174- 183.
40. Shneiderman, B. (1975) Cognitive Psychology and Programming Language Design, *ACM SIGPLAN,* 46-47.
41. Simon, Fincher,S., Robins, A., Baker, B., Box,I., Cutts, Q., De Raadt, M., Haden, P., Hamer, J., Hamilton, M., Lister, R., Petre, M., Sutton, K., Tolhurst, D., & Tutty, J. 2006. Predictors of success in a first programming course. In *Proceedings of the 8th Australasian Computing Education Conference (ACE'06)*.
42. Wang, Y., Patel, S. & Patel, D. 2006 A Layered Reference Model of the Brain (LRMB). *IEEE Transactions on Systems, Man, and Cybernetics, 36(2),* 124-133.
43. White, G.L. and M. P. Sivitanides (2002) A Theory of the Relationships between Cognitive Requirements of Computer Programming Languages and Programmers' Cognitive Characteristics, *Journal of Information Systems Education*, 13(1), 59-66.
44. WING, J.M. 2006. Computational thinking, *Communications of the ACM 49*, 3, 33–35.
45. WINSLOW, L. E. 1996. Programming pedagogy – A psychological overview. *ACM SIGCSE Bulletin 28*, 3, 17–25.
46. Xu, S. and V. Rajlich (2004) Cognitive Process during Program Debugging, Proceedings of *Third IEEE International Conference on Cognitive Informatics,* August, Victoria, 1-7.

IJERT