International Journal of Arts, Sciences and Education
https//ijase.org | ISSN: 1234-5678
Volume 1 Issue 1: 53-76

Understanding their Voices from Within: Difficulties and Code Comprehension of Life-Long Novice Programmers

BILLY S. JAVIER
Associate Professor, College of Information and Computing Science
Cagayan State University
billyjavier@csu.edu.ph
ResearcherID: V-6915-2017

Abstract: Programming is considered one of the grand challenges in computing education and attracts much attention in computing education research. Most areas of concerns were in to teaching methods, educational technologies, and student performance. This study uncovers the difficulties and code comprehension of novice programmers with the hope of providing efforts to computing institutions in improving programming competencies of novice programmers. Qualitative interview, observation, and document reviews were the methods in data collection towards understanding the voices within the thirteen (13) junior Information Technology life-long learners consented to participate in this mixed-method study. The participants were subjected to a hands-on experience, program tracing or debugging, and interviews along the process. Results showed the novice programmers' difficulties in program structures, code comprehension, program tracing or debugging, as well as code navigation. The results imply a growing concern for course designers in restructuring course methodologies providing real-world problem-solving cases and understanding the student's dilemma in writing, designing, code comprehension and tracing, as well as navigating the code. The current study was limited to understanding the difficulties and code comprehension among novice programmers, thus a future work to consider the teachers' difficulties and issues associated to novice programmers may be necessary.

Keywords: *Code Comprehension, Code Tracing, Programming difficulties, Novice Programmers, Programming*

I. INTRODUCTION

In this highly informative age, computer programming is becoming of equal and dire importance in practically every profession globally. Yet, programming is still problematic for students to learn especially for beginning students across all ages (Kelleher & Pausch, 2003). Problems in learning programming often hamper new computing students as novice programmers especially those without previous understanding and exposure to programming. In fact, this also stretches the patience and perseverance adding challenge to teachers (Krpan, Mladenović, & Rosić, 2015) (Phit-Huan Tan, 2009) teaching programming courses. One of the difficulties noted is to transform algorithm in mind into syntactical solution or source code (Yulianto, Prabowo, Kosala, & Hapsara, 2018). These errors in programming are highly

associated to the confusion of the program plan structure and semantic misunderstanding of language constructs (Ebrahimi, 1994).

Stirring at the most troublesome times and in surprising corners of students' system, programming errors can be very annoying and time consuming. In fact, frustrating experiences in programming contribute to undesirable learning outcomes and may lead to poor retention in the field (Ford & Parnin, 2015) (Essi Lahtinen, 2005). Typically, errors happen because the programmer may have failed to anticipate a detailed user action or behavior; made a certain error; forgot to check if the vital data to complete an action is missing or incorrect; may have used an external system that is offline or sends an incorrect data; employed libraries such as program types built into the operating system that are outdated or not present on the computer or server where the software is installed; or operated on a tasks that may have been outside their capability.

Programming is the act of assembling a convention of symbols and structures representing computational actions. Using these symbols, students define their intentions to the computer and given a set of symbols, a user who recognizes the symbols can foresee the performance of the computer (Kelleher & Pausch, 2003). Novice programmer in this context refer to any person, who is a beginner in programming (Sevella, Lee, & Yang, 2013). In this study, junior students who have knowledge about C programming which is one of the courses in their preceding semester are considered to be novice programmers (Sevella, Lee, & Yang, 2013). Novice programmers in universities find learning how to program problematic due to simultaneous learning of syntax, semantic or programming as well as how to interpret error messages. Many studies have been conducted to determine performances of novice programmers in writing programs. However, very few studies underscored common difficulties and code comprehensions among novice programmers especially in the Philippines. In recent years, a multi-national study showed that students have problems in writing program codes (M. McCracken, 2001) (Vego, 2009).

Related Literatures

Problems in learning programming often hamper university freshmen students as novice programmers. The study of Yulianto, Prabowo, Kosala, and Hapsara in 2018 highlighted the provision of a proposed application that improved the passing rate of novice programmers in learning programming towards transforming algorithm in mind to syntactical

solution. The use of visualization has been advocated as Balabat and Rojo(2012) revealed a significant improvement in the academic performances of novice programmers. While most findings proved a considerate improvement in learning programming on the use of visualizations, most novice programmers are not explicitly aware of the problem-solving process used to approach programming problems and cannot articulate to an instructor where they in that process (Prather, et al., 2018). As cited by Shuhidan, Hamilton, & D'Souza (2011), a Garner research revealed that there are 3 obstacles or difficulties in learning programming. Primarily, errors in writing syntax including curly braces and semicolon. Secondly is the difficulties in understanding and designing a program. Lastly is the difficulties in understanding the basic structure of a program. The same research findings was also confirmed in the study of Layona, Yulianto, and Tunardi in 2017 underscoring these difficulties.

Writing syntax is dependent on the programming language structures prescribed. In programming, syntax refers to the rules that specify the correct combined sequence of symbols that can be used to form a correctly structured program using a given programming language (Krpan, Mladenović, & Rosić, 2015). Programmers communicate with computers through the correctly structured syntax, semantics, and grammer of a programming language. However, students in introductory programming courses tend to a have a large number of syntax errors (Moore, Parrish, & Cordes, 1997). Typically these include simple typo-errors, while others are the results of some type of misunderstanding about the fundamentals of programming or the programming language itself.

Code Comprehension is considered as an essential part learning and creating programs in various programming concepts. It is considered as one of the most critical and time-consuming task during the actual programming and even in software maintenance process (Al-Saiyd, 2017). An empirical study into the use of software metrics as a way of estimating the difficulty of code comprehension tasks indicate that software metrics can provide useful information about code tracing difficulties in first year programming assessment (Kasto & Whalley, 2013). Code comprehesion problems have been shown to effective assessment items in computer science education. Also known as semantic analysis, code comprehension is the task of ensuring that the declarations and statements of a program are semantically correct, that is, their meaning is clear and consistent with the way in which program control structures and data types are supposed to be used.

Various studies had unlocked the teaching methods, educational technologies, and students performance in programming, yet very limited studies in the Philippine context were documented along difficulties of Filipino life-long learners were documented. While there may be multi-national studies covering the difficulties and issues of novice programmers, this study birdging the afore-mentioned gap seeks to uncover the difficulties in producing an executable program of junior information technology life-long learners in the Philippines.

Research question: This study aimed at unlocking programming difficulties of novice programmers. Specifically, the study determined the common difficulties of novice programmers in producing an executable program in terms of

Program structures used

Semantic Analysis or Code Comprehension

Program Tracing or Debugging

Code navigation

While students' performance in fundamental programming courses relates with producing quality capstone project (Ikonen & Kurhila, 2009) (Javier B. S., 2017), the study generally provided inputs for instructional development and methodological activities alongside teaching and learning programming in computing courses.

II. METHODOLOGY

Research Design

A mixed-method research design was utilized in the study imploring both quantitative and qualitative methods. Qualitative research is a common method for discovering and analysis of the meaning of individuals or groups of participants attributed to a certain societal or human issue. This involves developing questions and techniques, data typically gathered in the participant's locale, data analysis inductively arising from specific to general themes, and the researcher's capability to deriving interpretations of the meaning of the data obtained (Creswell, 2013). Having been engaged in this form of inquiry, the research is supported with the inductive way of analyzing the results of this study, concentrating on the participant's personal meaning of their programming difficulties and success factors, and the importance of doing problem-solving activities in programming courses for novice programmers.

More so, phenomenological research as described by is a design of philosophical and psychological investigation in which the researcher describes the lived experiences of individuals about a phenomenon as described by participants. (Giorgi, 2009) This description being applied in the study highlighted the essence of the experiences of these junior information technology life-long novice programmers.

Qualitative interview and document reviews were the methods to be used in data collection. Qualitative interview is used to "investigate varieties of human experience". They attempt to understand the world from the subjects' points of view and to unfold the meaning of their lived world" according to Kvale, 2006, p. 481 as cited by (Ferraris, 2014). A qualitative interview is more conversational and so will allow the participants to freely express their opinions on the subject matter. This methods works well with the at hand because because it aims to bring to bring out the interpretations and meaning made by the participants.

Sources of Data

The participants (key informants) of the study were selected through purposive sampling. Purposive sampling is a non-probability sampling method and it occurs when elements selected for the sample are chosen by the judgment of the researcher (Black, 2010). In this study, 43 junior and life-long learners taking up Information Technology were asked to participate in this study. The objectives, significance, manner of data collection, as well as specifics were first discussed upon approved consent from the College Dean. Initial consideration for selection is their completion of basic programming courses as evidenced by the academic performance requested and were currently taking regular semester load. Thirty-eight students were purposively listed as initial participants. However, from the 38, only 13 voluntarily participated in the study upon acceptance of the signed consent form. Formal consent from the participants were obtained to be included in the study. The study focused on the life-long learners currently enrolled at Cagayan State University as an effect of the implementation of the DepEd's K-12 Transition Program where no qualified sophomore, intended to be participants for the study, was enrolled.

Data Collection and Analysis

Interviews allowed for a deeper insight into the actual experience of the participants of this study. This also provided an avenue and opportunity to clarify themes (Merriam, 1998)

(Neuman, 2006). Interviews are good ways of checking the understanding of the researchers but it not necessary to the precise words used by the participants.

The gathering of information or data from the participants of this study involved an in-depth interview using the guide questions (see Appendix B – Interview Guide Question) which were recorded with permission (see Appendix C – Participants' Consent Form). Series of interviews with the participants were done in February to April 2019. The recorded interviews were then transcribed by the researcher while immersing with the information at hand. Another reason for doing the transcription ensures that the transcript was accurate and complete as possible. Coding system was implemented for the confidentiality of the identity of the participants. To ensure that the transcriptions were correct, valid and reliable, the researcher individually meet again with the participants to verify the transcripts. The participants were asked to give comments and suggestions about the transcripts. The last section of the interview transcripts affixed the participant's signature affirming the details of interviews made.

For the purpose of understanding the difficulties of novice programmers, the participants were asked to solve a particular problem in 3 laboratory hours. Programming teachers were solicited of their inputs and confirmation of the problem given in the specified duration, expected and applicable program structures used, and coverage of the learning content taught to the students. The case problem imposes the use of conditional structure, looping structure, procedures or functions, and single-dimensional array. The participants were then allowed to use C# programming language known to them and previously taught to them. The researcher asked their current subject teacher to do the actual tasks while observing the participants without intervening and causing disruptions. The researcher observed the class without intervening and instigating interferences. Outputs of each participating novice programmer were carefully checked and analyzed according to specifications set. After the hands-on experience, the participants were asked to participate in a consented interview to elicit from them their common difficulties as novice programmers in producing an executable program. Each participant was interviewed of their experience and difficulties as novice programmers. Their responses were then transcribed, analyzed, and interpreted. In another set of interviews, the participants were asked to trace or debug the complete program in a whole sheet. Follow-up questions were then made to elicit their problems and issues after having corrected the answers they have traced. Another round of interviews was individually

concluded to verify the transcripts transcribed by the researcher, affirming therein their responses made, and affixing their signatures.

Responses were described thru frequency counts, percentages, and ranks. The thematic analysis was used in analyzing the information gathered. Creswell (2013) declares that phenomenological data analysis proceeds through the methodology of reduction, the analysis of specific statements, and a search for all possible meanings. The participants' descriptions of experiences were clustered into statements. The theme of each experience was found out and were being clustered according to meaning. To make a general description of the participants' experiences, the common themes were arranged together.

III. RESULTS AND DISCUSSION

Most of the participants are male (8) outnumbering their 5 female counterparts, with age as reported ranging from 22-24, with access to computers both in school and at home (laptops), with access to free Internet connections, and had a satisfactory computed grade in the 2 programming subjects ranging from 86-90.

On **Common Program Structures Used**. The participants consumed the 3-hours hands-on experience to solve the problem given. After reviewing the code structures, majority of the respondents used mostly condition structures if, nested-if, and if…else structure, as depicted in Table 1, to solve the problem given.

Table 1: Program Structure Commonly Used

Programs Structure Commonly Used	f (n=13)	%	Rank
if structure or nested if structure	4 (M=1; F=3)	30.77	1
if…else structure or nested if…else	4 (M=2; F=2)	30.77	1
for structure	3 (M=3; F=0)	23.08	3
do…while structure	1 (M=1; F=0)	7.69	4
Array (one-dimensional)	0	0	6
Procedure or Functions	1 (M=1; F=0)	7.69	4

Carefully analyzing the participants' responses, most of the novice programmers utilized the selection structure over looping, arrays, and procedures. This finding suggest that more novice programmers tends to write programs basically using if or switch structure which they are most affluent to. Writing programs in selection structure are long and tedious, and

could not perform repetitive tasks. Further, writing without dividing into pieces affects the optimal performance of the program compilation including its execution.

In fact, most of the participants conveyed that they used those structures they are familiar and able to understand and apply in their programs. One participant expressed,

> *"Applying loops and functions is very challenging that it frustrates me even I exerted efforts reading some examples of those."*

A common keyword from the participants – "stressful" tends to suggest their dismay in not performing well in their programming subjects, affecting their academic performance. The novice programmers were mostly challenged in the application of the most applicable structure in a certain problem task. This finding agrees with study of Ford & Parnin (2015) relating to the stress and frustrations of the novice programmers in writing programs.

Digging up details why these structures were basically used and the foregoing were their transcripts:

> "*Honestly, mahaba yung code ko kasi I only used if...else structure. Hindi ko na kasi alam iimplement yung looping sa loob ng array."* [Honestly, I have a lengthy code since I only used if…else structure. I don't know how to implement loops in arrays.]
>
> *Pareho lang naman ung output, I see no problem in it. Nakakalito yung pag-gamit ng loop structures lalo na pag nasa function."* [It arrived with the same output, so I see no problem in it. It's confusing to use loop structures especially if within functions.]
>
> *"Hirap akong gawin kahit ung one-dimensional array. Mas ginamit ko yung nested...if kasi madali lng."* [I found difficulty to use even one-dimensional array in my programs. I preferred using nested…if because it is easy.]
>
> *"Laging may error akong naeencounter every time na ittry kong gawin ung program with array and function, kaya minabuti kong gumamit ng for at if structure."* [I encounter errors every time I tried to write programs in array and function, so I decided to use for and if structures.]
>
> *"Sa loop structure, for lang ang alam ko. Di ko na alam kelan gagamitin ang while sa do…while. Mas lalong hirap akong gamitin iyon sa function lalo na kung may parameter passing."* [I only know for among loop structures. I don't know when to use a while or a do…while. I find it harder to implement those in functions requiring parameter passing.]

From the transcripts it could be gleaned that life-long novice programmers tends to use basic structures due to their clear understanding of its use. More so, their difficulties in using advanced structures could be associated to their problem-solving and understanding of the key concepts of the programming constructs.

International Journal of Arts, Sciences and Education
https//ijase.org | ISSN: 1234-5678
Volume 1 Issue 1: 53-76

For the purpose of clarity, the prior programming subjects of the participants covered basic concepts, control structures (selection or conditional, branching, looping or iteration), operations (arithmetic, logical, relational), functions, arrays (one and multi-dimensional), procedures, sorting, and list. From the results, it maybe suggested the dire need for the novice programmers to learn more concepts relative to loops, arrays, and procedure and functions. Actually, the participants also conveyed teaching-related factors affecting their use of program structure to solve the problem given. A female participant conveyed,

> "*Madadali yung mga example during sessions, pero kapag actual problem solving na ang hirap iapply yung loops and arrays"* [Easy example were presented clearly during sessions, however on actual problem-solving cases, we find difficulties in applying loops and arrays].
>
> *"I find it difficult to comprehend what sir J*** explains about the flow of the program. It's different when I ask from my classmates. Madali na kumplikado, di ko ma-gets si sir"* [I find it difficult to comprehend what my teacher explains about the flow of the program. It is different from what my classmate explains. I find it easy yet complicated. I don't get what my teacher taught us].

From the above statements, the need for teachers to clearly explain for the students would be important to improve the understanding of students in computer programming. The study of M. McCracken (2001) proved the role of teaching in advancing the learning outcomes among novice programers which are eventually software developers in the later years.

On Semantic Analysis or Code Comprehension. In the semantic analysis or code comprehension of the programs they write, participants expressed their difficulties along understanding programming structures especially when applied in their programs. The copy of the program created were presented to the participant during the interview. They were asked how do they comprehend on the structures used. They were then asked on which structures they hardly comprehend as novice programmers. Table 2 presents the different structures where novice programmers hardly comprehend.

Table 2: Structures where novice programmers hardly comprehend

	F (n=13*)	%	Rank
1. Selection structures	3	23.08	7
2. Branching structures	9	69.23	3
3. Looping structures	10	76.92	2
4. Arrays	13	100.00	1

5. Procedures and functions	9	69.23	3
6. Arithmetic or mathematical structure	4	30.77	5
7. Logical structures	4	30.77	5
8. Relational structures	3	23.08	7

*multiple occurrence

Analyzing both the responses in the interview with their coded programs as well as based on their experiences in the previous programming subjects, most of the novice programmers expressed their difficulties along arrays (100.0%), looping (76.92%), and procedures and functions (69.23%). While a few participants conveyed their difficulties in comprehending logical structures, arithmetic, selection and relational structures, this aggravates the issue along creating programs or writing codes for solving problems since these structures are fundamentals of programming which greatly affects their comprehension and writing of programs applying advanced programming concepts and structures. In the study of Balmes (2017), mathematical ability relates to programming ability, hence, the lack of comprehension skills even in arithmetic and logical structure would tend to cause more difficulties in understanding other structures.

In fact, during the 2nd round of interview with the participants, most of the novice programmers stated that they failed to recognize the need to integrate logical-thinking and critical-thinking into real-world problem-solving activities especially integrating those with looping structures and in procedures and functions.

> *"Hirap ako mag-program, dahil hindi ko maintindihan yung combinations ng structure. Hindi ko napagsasama-sama yung required na structure dun sa program na ginawa ko".* [I am hard up doing a complete program because I could hardly comprehend combinations of structure in the program. I cannot integrate all required structure in the program I am writing.]

From the interviews, the following transcripts affirmed the structures that novice programmers hardly comprehend,

> *"Kahit maraming examples si teacher, pagdating sa actual hands-on programming test, ang kumplikado na nung gagawin lalo na kapag arrays at loops lang required."* [Though more examples are given, once on actual hands-on programming tests, the programming tasks becomes more complicated especially when arrays and loops are the only required structures.]

> *"Hindi ko maintindihan lalo ung matrix ng 2-dimensional array. Mas wala akong nagagawa sa lab. Pag inexplain naman siya parang madali lang."*

[I hardly comprehend especially matrices of 2-dimensional array. I cannot do anything in the laboratory. It's so easy when clearly explained.]

"The use of switch with operations made me sick! Sa dami ng operations na pinagsasama sama mo sa problem solving scenario, parang gusto ko nang magquit agad!" [The use of switch structure with (arithmetic, relational and logical) operations made me sick! With so many operations when altogether used in a problem-solving scenario, I wanted to quit already!]

May mga operators na parang madali sa case problems, pero kapag sinimulan mo siyang gamitin kahit sa if structure lang, nagiging complicated na. I don't know where to start! [There are operators that applies easy in case problems, but when coupled with even just an if structure, it's going to be complicated. I don't know where to start!]

Sa mga tracing ng array and functions of real-case scenarios, parang wala ni isang tumama sa mga ginawa ko, nakakahiya parang wala man lang ako natutunan. [Tracing and writing programs in array and functions of real-case scenarios, I think not even one from my outputs were correct, I feel sorry, I think I haven't learned anything in class.]

Life-long novice programmers tends to find difficulty in comprehension of both basic and advanced structures. This tends to suggest the need for restructuring teaching and learning how to better understand programming structures from program drills by implementing more or real-world case scenario.

The difficulty in understanding programming structures according to Essi Lahtinen (2005), seems to be that the student overestimate their understanding, that the biggest problem of novice programmers does not seem to be the understanding of the basic concepts but rather learning to apply them. Similarly, Kelleher & Pausch (2003) also disclosed the same findings where most novice programmers hardly comprehend on structures when applied in real-world problem solving cases.

On Program Tracing or Debugging. The study has asked the participants to trace a validated program or code snippet leading to identifying the program output when compiled and translated or executed.

Asking the participants how do they feel they performed, sharing their views or issues in tracing or debugging the code snippets given, the participants said

"Madali lang yung mga simpleng if conditions. Tiwala akong nagawa at nakuha ko ng tama yung sagot. Medyo nahirapan lng ako dun sa part na may array." [I find it easy tracing the simple if structures. I am confident that I got it correct. However, I find it difficult along that part with array structures.]

"Nalito ako sa loops na nakaembed sa for structure. Di ako sure dun sa resulta if nakuha ko siya kahit binigay ko na ang best ko." [I got confused with the loops embedded in the for structure. I am not sure with the result if I got it correctly though I gave my best.]

"I think nagawa ko ng tama yung mga switch at for structure part. Pero dun sa part na may array feeling ko di ko naibigay ang best ko. Hirap akong intindihin at iextract yung correct results." [I think I have done correctly with the switch and for structure part. However, I feel I did not make best along array parts of the program. I find difficulties understanding and extracting the correct results of the program.]

"Dun sa part na locating the program errors or error debugging, nahirapan ako dun sa syntax part. Sigurado akong 5 out of 10 lang ang nahanap ko at nacorrect ko dun. Tapos kung itrace mo na yung results, alanganin ko sa may functions and array part. Makapabagtit!" [I find issues or difficulties in the syntax part when locating the program errors or error debugging. I'm pretty sure I got 5 out of 10 correctly. Upon tracing the program output, I feel worried along functions and array part.]

Table 3 summarizes the participants performances on program tracing or debugging.

Table 3: Program Tracing Performance of Novice Programmers

Program Tracing Performance	**f (n=13)**	**%**
Excellent (Completely traced the exact and correct output)	1 (M=1; F=0)	7.69
Very Satisfactory (Completely traced correctly but with less than 5 errors in the output)	2 (M=1; F=1)	15.38
Satisfactory (Completely traced but with less than 7 errors in the output)	5 (M=4; F=1)	38.46
Fair (75% traced with less than 10 errors in the output)	4 (M=2; F=2)	30.77
Poor (75% traced with more than 10 errors in the output)	1(M=0; F=1)	7.69

While most of the novice programmers (38.46%) fairly performed in the program tracing or debugging, it is noteworthy to reveal the excellent performance of one male novice programmer. The program tracing requires the skills to determine and extract the output of a given program structure or code snippet (Bennedsen & Caspersen, 2012). From the interviews,

most of the issues made known includes (a) confusing traceability of the code snippet due to the program structures used; (b) looping structure used embeds a conditional structure that causes them misunderstanding; (c) difficulty in the logical flow of the code snippet; and (d) misunderstanding of the output structure needed to be extracted.

> One novice programmer recalled,
> *"Yung mga code samples ay parang andali dali kapag tinignan, pero kumplikado na pala kapag ini-isa-isa mong itrace dahil sa mga conditions ata ginamit na looping statements."* [The code snippet was as easy as you see, but it's complicated as you trace the lines of code because of the loop and the condition statements.]

In fact, visualization may have helped novice programmers improve learning programming concepts as mentioned by Balabat & Rojo (2012) but understanding the code and debugging them line-per-line would require skill in understanding the flow and the general meaning of the program created (Prather, et al., 2018).

With the follow-up questions on which part or parts of the tracing program test case the participants were problematic, the following were some of their responses:

> *Yung part na may loop structure sa loob ng nested…if structure. Kasi nakakalito na yung condition and increment values nya.*
>
> *Doon sa part that the value is passed through a function or another, Hindi ko na alam anong naipasa.* [When I reached that part that the value is passed (passing parameter value) from a function to another. I don't know what value needs to be passed (returned)].
>
> *"The values from the array variables. Nawala ako dun sa index niya mula sa for loop. It's very frustrating, sayang."* [The values from the array variable. I got lost in the index value (indices value) based on the for-loop value. It's very frustrating.]
>
> *Nahirapan akong intindihan mula dun sa declared variable na may initial value at connected siya dun sa next variable na may formula. Tapos nagbabago ito kapag tinawag na sa loob ng for structure.* [I find difficulty in understanding beginning from the declared variable with initial value and was used with the next variable containing a formula, then the value changes each time it was called in the for-loop structure.]
>
> *Madali lang yung part na may if structure, kahit pa samahan iyon ng logical at relational expression. Pero kapag nasa function na tumawag kay array, hindi ko na alam itrace.* [It is easy along that part containing if structure, even if

it is used within a logical and relational expression. However, if it is in a function calling for an array variable, I don't know how to trace the program.]

The findings suggest that novice programmers tend to perform better in programming once they are able to comprehend on what the code means and knows how to trace the code snippets line-per-line. Tracing the code on a computer screen maybe tedious especially when errors in the program becomes sophisticated (Ford & Parnin, 2015).

On code navigation. Observing the programming activities of the novice programmers, navigating the lines code, as presented in table 4, illustrated their issues and concerns.

Table 4: Code Navigation Issues Experienced

Code Navigation Issues Experienced	f(n=13)	%	Rank
1. Highlighted line has no syntax issue	4 (M=2, F=2)	30.77	4
2. Running program but error occurs, no explicit error message appeared	4 (M=2, F=2)	30.77	4
3. Longer lines of codes, longer time to navigate (time-consuming)	8 (M=3, F=5)	61.54	2
4. Goto and/or jump to the error line identified not available (interface issue)	2 (M=1, F=1)	15.38	6
5. Navigation of the code by "random mutation" sometimes for hours on end	10 (M=4, F=6)	76.92	1
6. Read line – per – line	5 (M=3, F=2)	38.46	3

*multiple response

As presented, a substantial percentage (76.92%) disclosed the code navigation by "random mutation" sometimes for hours on end, while some preferred to read each line of code line-per-line. The findings tend to reveal that majority of the novice programmers find difficulties in navigating their own program. This may be attributed to their use of selection structures which requires longer lines of code, and eventually affecting the compiling time of the codes. There were advantages and disadvantages of the code navigation practices. First, Lister (2007) proved that students who attempt to debug code, sometimes for hours on end, by "random mutation", explains some of the more bizarre moments in teaching programming to novices. Second, finding errors thru code navigation from own program becomes a difficulty or challenge to novice programs (Essi Lahtinen, 2005). At the light side, reading line-per-line

may be frustrating (Ebrahimi, 1994) but would improve code comprehension and uncovering new insights from the written program.

In the interview, most of the participants said they hardly navigate longer lines of code, which tends to mean they need to write correctly the right syntax applying the best semantic analysis for the program. Some of the participants said,

> *"Matrabaho ang pag-navigate lalo na kung masyadong mahaba yung program codes. Madalas kung anong itinuturo ng error line, hindi un ang mismong error kundi ung mga nasa itaas nito"* [Navigating the code in a program especially with longer lines of codes requires time and effort. Most of the time, what the error highlighted is not the error, but those located before the highlighted code segment.]

> *"Minsan yung errors na tinutukoy eh hindi naman logically erroneous, kaya nawawala ako. Kapag itry mong ilagay yung suggested options, kumonti yung error pero pag inirun mo na siya walang output na matino."* [Sometimes, those errors highlighted is not logically erroneous which adds confusion to me. If you try to insert the suggested options, there appears lesser error messages, but when executed, there is no rightful output presented.]

One approach to high-pressured situations when everything goes wrong is code review or going over line-by-line, as conveyed by a male novice programmer,

> "*I need to go back to the code or syntax and go over line-by-line just to get the correct output.*" [I needed to go back to the codes written or syntax used and go over line-by-line the codes just to get the correct output.]

The difficulties of novice programmers along code navigation also agree with the results of (Yulianto, Prabowo, Kosala, & Hapsara, 2018) and (Ford & Parnin, 2015) which emphasized the role of the programming environment or the software to aid novice programmer in doing, tracing, and debugging their own written programs.

On Perceived issues or difficulties in terms of Course Contents. The participants were asked to state their perception on the course contents in programming as novice programmers. Table 5 presented the kind of issues novice programmers feel difficult in learning programming.

Table 5: Perceived Issues or Difficulties in Learning Programming

What kind of issues do you feel difficult in learning programming?	f(n=13)	%	Rank

1.	Using program development environment	2 (M=1; F=1)	15.38	6
2.	Gaining access to computers or networks	2(M=1; F=1)	15.38	6
3.	Understanding programming structures	8 (M=4; F=4)	61.54	3
4.	Learning the programming structures	4 (M=1; F=3)	30.77	5
5.	Designing the program to solve a certain task	10 (M=3; F=7)	76.92	1
6.	Dividing functionality into procedure	10 (M=5; F=5)	76.92	1
7.	Finding bugs from my own program	8 (M=3; F=5)	61.54	3

In table 5, a substantial number of the novice programmers disclosed their most difficult issues in programming was along designing the program to solve a certain task (76.92) especially among females, as well as dividing functionality into procedure (76.92) both among male and female novice programmers. Other issues underscored understanding programming structures (61.54%) and finding bugs from my program (8 participants) especially among females.

The findings suggest that novice programmers with their perceived issues and concerns along course contents in the programming subjects, there is a need to hasten and improve their understanding of the case problem given considering designing of program codes in procedures and functions. These require understanding larger entities of the program instead of just details, as also found in the study of Soloway and Spohrer (1989) as cited by Essi Lahtinen (2005).

In the interviews made, most of the participants agree having difficulties in the programming concepts such as input/output handling, error handling, using libraries, parameters in functions, and abstract data types. One female novice shared her thoughts about her difficulties in learning these course contents:

> *"Andami daming structures na pinepresent at lalo akong nahihirapan i-absorb at gamitin ito sa programs ko, lalo na kung paano ihandle ang errors sa program at pagpasa ng values sa functions"* [A lot of programming structures were presented to us in sessions and I am more challenged to absorb and implement those in my programs, especially on error handling and passing values in functions.]

> *"Basta gumana ung program ko okay na ako. Kaso if required na may functions or sub-program, hirap na hirap na ako intidihin at gawin ung desired output. Nakadagdag pa yung ilalagay mo sa array."* [Once the program work, I'm okay with it. However, if functions or sub-program is required, I find it

difficult to understand and work for the desired output. Including those with array adds up to my difficulties.]

"Masaya na mahirap aralin ang programming. Masaya lalo na kapag nasundan mo yung processes at walang errors yung output ko. Mahirap naman kapag hahanapin mo yung cause ng bugs or errors at ung pagsasama sama ng control structures." [Learning programming is fun yet difficult. It's fun especially when you are able to follow the processes and no errors are seen in my output. I find it difficult when finding the bugs or errors in the program, and when control structures are integrated.]

"Once complex case problem needs to be programmed, yung atake kung paano yung program flow at kung paano isosolve yung bawat task niya, dun ako nagkakaproblema. Basically, dapat coordinated ang inputs sa processes para magawa ng program ung desired output." [Once a complex case problem needs to be programmed [via a programming language], how to attack with the program flow and how to solve each task, frustrates me most. Basically, it should have coordinated inputs and process flow in order to define the program for the desired output.]

On learning and teaching programming. The novice programmers were enquired when do they feel that they learn issues about programming. Table 6 presented the results.

Table 5: Novice Programmers Issues on Learning and Teaching Programming

When do you feel that you learn issues about programming?	f(n=13)	%	Rank
1. In lectures	3 (M=1; F=2)	23.08	4
2. In exercise session in small groups	4(M=2; F=2)	30.77	3
3. In practical sessions	8 (M=3; F=5)	61.54	2
4. While studying alone	4 (M=2; F=2)	30.77	3
5. While working alone on programming coursework	11 (M=3; F=8)	84.62	1

Most novice programmers felt that they learned issues about programming while working alone on programming coursework (84.62%), in practical sessions (61.54%), and while studying along or in exercises session in small groups. While participants disclosed that they learn issues about programming in lectures (23.08%), the findings suggest that novice programmers tend to find difficulties or issues in programming after their lectures or after the class. While enrichment activities in this case are made, the novice programmers tend to encounter more issues about programming outside the class. The results agree with a study relating to the need for teaching and learning to be more comprehensive and real-world based program examples need to be made (Krpan, Mladenović, & Rosić, 2015) .

"Nasusundan ko naman kapag nasa class at ineexplain ni sir, pero kapag kami na nga barkada ang gagawa, dun nagsisimula yung problema. Lalo na kapag take-home at isa-isa na gagawa" [I am confident about learning how to do the program and comprehend when explained in class by my teacher. However, when with small group or friends, the issues and difficulties arise, most especially when assignments are made and individual coursework is expected].

"Nagkakaproblema akong matutunan ang programming kapag ako na mismo ang gagawa ng coursework namin, lalo na if individual ang presentation. Sa klase parang andali dali. Sa laboratory session, pag naibigay na yung sample at gagawa ng isang program na, hirap na akong buuin at makapagproduce ng maayos na program." [I find issues learning programming once a programming coursework needs to be prepared and presented individually. It seemed very easy in classes. During practical laboratory sessions, once a sample is given and we are required to write a program, I am hard up completing/producing a proper and executable program.]

"Yung mga classmates ko, kapag naiexplain na sa lecture namin at group exercise madali lang, kaso hirap na akong gumawa ng sarili kong program sa laboratory class namin especially kung may higher level ng program structure." [My classmates find it easy doing their coursework once explained in lectures and group exercises, yet I find it hard to do one working program in our laboratory class especially integrating with higher program structures.]

From the responses, most of the participants finds ease of learning and teaching programming in lectures and group exercises. Hence, this reflects the teaching and learning styles of both the students and teachers. However, with their experiences working alone on programming coursework they most felt those issues about programming. The experiences of novice programmers were actually from them especially when doing the program on their own (Vego, 2009).

On Learning Materials helpful in learning programming. The participants in the study were requested to share their helpful learning materials in programming.

When asked about which material helpful in learning programming, the participants shared their views as follows:

"Gustong-gusto ko yung isa-isang ineexplain ni sir yung sample program. Madalas yung sample na hanggang tatlo eh nakakapagbigay linaw sa programming topic naming." [I really wanted those sample programs each explained by my teacher. With 3 sample programs, I find clarity learning our programming topic.]

International Journal of Arts, Sciences and Education
https//ijase.org | ISSN: 1234-5678
Volume 1 Issue 1: 53-76

"Since naturuhan na kami ng syntaxes ng structures, mas nakakatulong sa akin yung Q and A part mula sa programming samples. Hindi lang yung basta nakinig at naitindihan mo yung sinasabi ng teacher. So, I find learning programming thru asking questions." [Since we were taught already with the syntaxes of structures, doing a question and answer from programming samples is of great help, not only just listening and understanding what the teacher explains. So, I find learning programming thru asking questions.]

"With the proliferation of Internet using mobile phones, mas prefer naming magself-study via free tutorials sa Internet. Medyo nakakahiya kasing laging nagtatanong ka sa klase. Siyempre ok na yung sa klase pero mas nagaadvance na ako via free tutorials online." [With the proliferation of Internet using mobile phones, we prefer to have self-study via free Internet tutorials. Always asking questions in class may be annoying. Discussions in class is okay but I make advance studying via free tutorials online.]

"Yung gumamit si sir ng diagrams para mavisualize yung flow ng program, I think effective yun, kasi kadalasan tamad na magbasa ng text ang mga IT students." [The use of diagrams by my teacher to visualize the flow of the program is think effective because most of the time IT students are lazy reading text (long lines of codes within programs).]

"Sa laboratory exercises at coursework, may gusto ko ng drag-drop or objects kasi nakikita mo na yung ginagawa mo kahit hindi pa nairun, kesa yung purely coding na medyo mahaba at di ka pa sure na gagana. Effective yung diagrams sa klase. [In our laboratory exercises and course works, I loved using drag-drop of objects because you can already see (visualize) what your program looks like even not in executable mode, compared to purely coding that is lengthy and you are not sure if it is working properly. Diagrams are effective in (teaching) the class.]

Collating all responses of the participants, table 7 summarizes their kind of materials that have helped or would help participants in learning programming.

Table 7: Materials Helpful in Learning Programming

What kind of materials have helped/would help you in learning programming?	f(n=13) *	%	Rank
1. Programming coursebooks	4 (M=0; F=4)	30.77	5
2. Lecture notes of copies of the presentation	2(M=1; F=1)	15.38	7
3. Exercise questions and answers	8 (M=4; F=4)	61.54	3
4. Example programs	10 (M=3; F=7)	76.92	1
5. Still pictures of programming structures	4 (M=1; F=3)	30.77	5
6. Interactive Visualization	10 (M=5; F=5)	76.92	1
7. Internet (free tutorials)	8 (M=3; F=5)	61.54	3

*multiple response

Most novice programmers prefer example programs and interactive visualizations (76.92%) respectively as helpful learning materials in programming. With the advent of the free Internet, participants also prefer free Internet tutorials, as well as obtaining exercise questions and answers (61.54%) respectively, over coursebooks, still pictures, and lecture notes. In fact, visualization may help improve understanding by novice programmers as proved by (Balabat & Rojo, 2012) as well as provision of more relevant example programs to novice programmers as suggested by (Layona, Yulianto, & Tunardi, 2017).

The finding tends to suggest that there are varied means of improving learning programming among novice programmers via interactive visualizations, internet tutorials, and more relevant example programs. This would mean that teachers should be giving ample relevant program example to novice programs while underscoring their semantics and proper syntax to avoid program difficulties and issues.

On Perceived Factors that leads to Poor Performance in Programming. The participants were asked of the contributing factors that may lead to poor performance in programming among novice programmers. These factors were presented in table 8.

Table 8: Perceived Factors that Leads to Poor Performance in Programming

What do you consider as factor that leads to poor performance in programming as novice programmers?	f(n=13)	%	Rank
1. Less examples on practical use shown	5 (M=2; F=3)	38.46	3
2. Not fully-functional computers provided in the laboratory	2(M=1; F=1)	15.38	7
3. Less effective teaching methodology	4 (M=3; F=1)	30.77	5
4. Quality of Presentation of the teacher	4 (M=1; F=3)	30.77	5
5. Theory-focused syllabus	8 (M=3; F=5)	61.54	2
6. Too wide coverage of the syllabus	10 (M=4; F=6)	76.92	1
7. Non-conducive learning environment	5(M=2; F=3)	38.46	3

*multiple occurrence

As expressed by the novice programmers, the greatest factor that leads to poor performance in programming was the wide coverage of the syllabus (76.92%), followed by a theory-focused syllabus (61.54%), and less examples in practical use shown as well as non-

conducive learning environment (38.46% respectively). This finding implies the need for the programming subject syllabus to consider specific topics for the programming contents, underscoring more of practical applications and less on theory, and providing more practical examples understood by novice programmers. The studies of Bennedsen & Caspersen, (2012) and Rane-Sharma, Sharma, Raman, & Sasikumar (2010) stressed out the focused of programming concepts and practical examples in programming subjects which greatly influence performance of student-programmers. One participant suggests

> *"Sana bawasan sa course syllabus namin yung madaming concepts, focus na lang sa mga practical structures na kelangan namin, at yung magagamit lng namin sa actual programming world"* [It is hoped that concepts in the course syllabus be minimized, focus on the practical structures that we need, and those we can actually use in the actual programming world.]

While instructional systems contribute to learning and performance of the students as emphasized in a study, (Javier B. S., 2015) novice programmers also underscored to considering the learning environment towards improving performance in programming.

IV. CONCLUSION AND RECOMMENDATIONS

From the findings, it is concluded that novice programmers find difficulties and issues particularly in understanding the program they create, test, and debug, and the structures they used in the program whether individually or in small groups or in class. Programming may be difficult but utilizing interactive visualization and provisions of relevant example programs to novice programmers may help improve learning programming.

Pedagogical Implications

The findings of this study could have ramifications in the academic service of the study site. It might want to consider the results in crafting an instructional delivery program on improving programming skill responsive to diversified skills acquisition, learning delivery mode especially on areas where the respondents need to get enlightened as well as advancing pedagogical bases highlighting helpful course materials.

Policy Recommendations

A policy document on a one-unit course programming readiness for computing students may be undertaken prior official enlistment to any programming course maximizing flexible

teaching and learning delivery methods. accommodating those disadvantaged ones, an online or offline intervention through modular approach but strictly monitored thru remote modality could be embarked.

In light of the foregoing findings, the researcher hereby recommends the following:

a. There is a need to restructure teaching and learning methodology to uncover difficulties of novice programmers in learning programming;
b. A more practical programming drills, and relevant examples must be made to ensure understanding and code comprehension among novice programmers;
c. A review of the course syllabus on programming courses may be considered to focus on the specific and effective strategies be anchored; and
d. Students who would venture on programming courses may be advised to enriched their arithmetic, logical, and critical problem-solving skills in order not to experience difficulties in writing programs.

Acknowledgement
The author expresses his profound gratitude to the administration of the Cagayan State University headed by Dr. Urdujah A. Tejada and Campus Executive Officer Dr. Simeon R. Rabanal, Jr. of the Aparri Campus for the kind approval in the participation of the author in the Diploma Program in Research Education of the Regional Center for Innovation in Teaching Excellence, Isabela State University. Further, the author expresses his deepest appreciation to the participants and cooperating faculty members of the College of Information and Computing Sciences, Cagayan State University at Aparri thru the Dean Dr. Corazon T. Talamayan. The author is grateful to the management staff of the ReCITE – ISU Cabagan for the mentoring and kind assistance towards the provision of this intellectual output. Finally, sincerest gratitude is warmly extended to the Commission on Higher Education for the grant extended to the author.

V. LITERATURE CITED

Al-Saiyd, N. A. (2017). Source code comprehension analysis in software maintenance. IEEExplorer. Retrieved March 2019, from https://ieeexplore.ieee.org

Balabat, C. A., & Rojo, J. N. (2012). A Program Visualization Approach in Developing an Interactive Simulation of Java Programs for Novice Programmers. *Mindanao Journal of Science and Technology*, 63-80. Retrieved December 2018

Balmes, I. L. (2017, July). Correlation of Mathematical Ability and Programming Ability of the Computer Science Students. *Asia PAcific Journal of Education, Arts, and Sciences, 4*(3), 85-88.

Black, K. (2010). *"BUSINESS STATISTICS: CONTEMPORARY DECISION MAKING"* (6th ed.). John Wiley and Sons.

Creswell, J. W. (2013). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (4th ed.). Thousand Oaks, California, US: SAGE Publications, Inc. Retrieved from https://www.academia.edu/29332705/John_W._Creswell_Research_Design_Qualitative_Quantitative_and_Mixed_Methods_Approaches_SAGE_Publications_Inc_2013_

Ebrahimi, A. (1994). Novice programmer errors: language constructs and plan composition. *International Journal of Human - Computer Studies, 41*(4), 457-480. doi:https://doi.org/10.1006/ijhc.1994.1069

Essi Lahtinen, K. A.-M.-M. (2005). A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin, 37*(3), 14-18. doi:10.1145/1067445.1067453

Ford, D., & Parnin, C. (2015). Exploring Causes of Frustration for Software Developers. *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, 115-116. doi:10.1109/chase.2015.19

Giorgi, A. (2009). *The descriptive phenomenological method in psychology: A modified Husserlian approach.* Pittsburgh, PA, US: Duquesne University Press.

Ikonen, M., & Kurhila, J. (2009). Discovering high-impact success factors in capstone software. *10th ACM conference in SIG-information technology education*, (pp. 235-244). doi:10.1145/1631728.1631803

Javier, B. S. (2015). Determinants of Employability of the Information Technology Graduates of Cagayan State University. *Countryside Development Research Journal, 3*(1), 43-52.

Javier, B. S. (2017, December). Jumpstarting 21st Century Capstone Projects: Pros and Challenges among Information Technology Students of Cagayan State University Philippines. (E. Iringan, Ed.) *3rd SPUP International Research Conference Journal, 3*(1), 71-78. Retrieved from www.researchgate.com

Kasto, N., & Whalley, J. (2013). Measuring the difficulty of code comprehension tasks using software metrics. *ACE '13 Proceedings of the Fifteenth Australasian Computing Education Conference. 136*, pp. 59-65. Adelaide Australia: Australasian Computer Society Inc. Retrieved December 2018

Kelleher, C., & Pausch, R. (2003). *Lowering the Barriers to Programming: A Survey of Programming Environments and Languages for Novice Programmers.* doi:1073-0516/01/0300-0034

Krpan, D., Mladenović, S., & Rosić, M. (2015, February 12). Undergraduate Programming Courses, Students' Perception and Success. *Procedia - Social and Behavior Sciences, 174*, 3868-3872. doi:https://doi.org/10.1016/j.sbspro.2015.01.1126

Layona, R., Yulianto, B., & Tunardi, Y. (2017). Authoring Tool for Interactive Video Content for Learning Programming. *Proc. Comput. Sci*, (pp. 37-44). doi:10.1016/j.procs/2017/10/006

Lister, R. (2007, January). *The Neglected Middle Novice Programmer: Reading and Writing Without Abstracting.* Retrieved from ResearchGate: https://www.researchgate.net/publication/228900648

M. McCracken, V. A. (2001). A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First Year Students. *ACM SGCSE Bulletin, 33*(4), 125-140. Retrieved February 2019

Merriam, S. (1998). *Qualitative Research and Case Study Applications in Education.* San Francisco: Jossey-Bass Publishers.

Moore, D., Parrish, A., & Cordes, D. (1997). Analyzing syntax error patters among novice programmers. *ACM-SE 35 Proceedings of the 35th Annual Southeast Regional Conference* (pp. 188-190). Murfreesboro, Tennessee: ACM. doi:10.1145/2817460.2817508

Neuman, W. (2006). *Social Research Methods: Qualitative and Quantitative Approaches.* Boston: Pearson.

Phit-Huan Tan, C.-Y. T.-W. (2009). Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception. *International Conference on Computer Technology and Development*. doi:10.1109/ICCTD.2009.188

Prather, J., Pettit, R., McMurry, K., Peters, A., Homer, J., & Cohen, M. (2018). Metcognitive Difficulties Faced by Novice Programmers. *Proceedings of the 2018 ACM Conference on International Computing Education Research* (pp. 41-50). Espoo, Finland: ACM New York. doi:10.1145/3230977.3230981

Sevella, P. K., Lee, Y., & Yang, J. (2013). Determining The Barriers Faced by Novice Programmers. *International Journal of Software Engineering (IJSE), 4*(1). Retrieved April 2019

Shuhidan, S. M., Hamilton, M., & D'Souza, D. (2011). Understanding Novice Programmer Difficulties via Guided Learning. *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science* (pp. 213-217). Darmstadt, Germany: ACM New York. doi:10.1145/1999747.1999808

Stein, M. (2003). Student effort in semester-long and condensed capstone project courses. *Journal of Computing Sciences in Colleges, 18*(2), 200-212.

Vego, J. (2009). Interest in CS as a major drops among incoming freshmen. *Computing Research News, 17*(3), 126-140. Retrieved March 2019

Yulianto, B., Prabowo, H., Kosala, R., & Hapsara, M. (2018, April 9). Novice Programmer = (Sourcecode) (Pseudocode) Algorithm. *Journal of Computer Science, 14*(4), 477-484. doi:10.3844/jcssp.2018.477.484