Integrated development environment for program logic formulation

May M. Garcia[1]*, Ma. Carmela F. Francisco[1]

[1]*Department of Mathematics, Technological University of the Philippines, Manila, Philippines*

ARTICLE INFORMATION

Article History:
Received 12 January 2015
Received in revised form 17 April 2015

*Corresponding author: lorzenmay@yahoo.com

ABSTRACT

Programmers translate the way people do to solve problems into programs. By doing so, good logic formulation is a must. The study aimed to develop an Integrated Development Environment (IDE) for Program Logic Formulation to enhance students' skills in programming through flowcharting.

The IDE was designed to include tools for creating and editing flowchart, checking program logic and expressions, debugging, and maintaining the database of programming problems. The IDE's interface was partitioned into several sized windows in which all tools are contained and labelled. The design used icons for flowchart symbols where any one of these can be clicked and dragged into the provided workspace, file functions such as new, save, save as, cut, copy, paste, delete, and print, and translator functions such as compile and debug. The translators in the developed system are the compile and debug modules. The compile module checks the expressions and operations on the flowchart symbols. If there is error found in the symbols, it produces one from any of the three messages: The flowchart is syntactically correct, the program produces the correct logic, and it produces wrong output. The debug module examines each symbol from start until end symbol in the flowchart. There are two programs running simultaneously: C# (.cs) file and the IDE itself. Behind the scene, the IDE creates a .cs file consisting of equivalent C# statements for each flowchart symbol and other C# statements to make the program complete for execution. Each C# output for process and output symbols is displayed and read on the console command and passes it back to the IDE's Debug window. This IDE was tested for its functionality and reliability to determine correct flowchart and logic.

Testing results showed that the IDE provides a working environment where students can create, save, open, edit, and print flowcharts as well as check program expression and logic. Based on the results of the evaluation conducted, the developed system gained an overall mean of 4.62 with a descriptive rating of Excellent, which indicates that the software material can be a useful tool in training and enhancing the students' skills in logic formulation.

Keywords
IDE, Program Logic Formulation, PLF, Flowcharting, Logic Formulation, Pseudocode

Introduction

Today's technology brings much advantage to humankind from industrial to personal activities. Industries use software tools as aids in gathering, analyzing, and interpreting data. Similarly, office personnel use productivity tools such as word processor, spreadsheet, and presentation software to help them in doing their job with ease and satisfaction. In the academe, teachers practice the use of instructional tools or courseware package to improve their delivery of instruction.

Some of the available coursewares are inclusive in terms of usage. The software, Packet Tracer of CISCO Company, helps students to design and check the computer connection in a Local Area Network (LAN) environment. It is used by most teachers handling Network subject. Also, the Pseudo-Compiler of Informatics, checks the syntax of the program written in the English language and later shows its output if correct. This software requires the students to be knowledgeable of the syntax customs in using Pseudo-Compiler. In this case, the students should be aware of the language structure of Pseudo-Compiler. However, this is not a commercially acceptable computer language in programming. Learning the Pseudo-Compiler of Informatics prolongs the time to develop students' skills in programming because they should be knowledgeable of flowcharting which is an important tool for them to understand programming.

Background of the Study

Writing a computer program is not as simple as one thinks for it must conform to the required logic–correct thinking and reasoning. For a person to know how to write a program, logic formulation is a prerequisite skill.

Different tools can be used to show the program logic such as *pseudocode*, *nassineiderman*, and flowchart. Flowchart is used to show the step-by-step process to solve a problem before it is translated into a program using any programming language. Some students find it difficult to learn programming in a period of one semester. One of the reasons is that they are not well guided in analyzing a problem and formulating its solution. Also, teachers give them limited exposure to computer programming exercises and they have not mastered the basic concepts on logic formulation. Some students cannot formulate the logic the way they think how to solve a problem possibly because they are not used to solving it alone. They are neither accustomed to do the desk checking method, where they themselves can check their own solutions. If students could be trained to simulate logic through flowchart, then learning programming would be an easy task.

Objectives of the Study

General Objective:

The general objective of the study is to develop a stand-alone IDE in program logic formulation intended for TUP students in the subject Computer Programming I.

Specific Objectives:

Mainly, this study has the following objectives:

1. Design a stand-alone system that draws flowchart with the following features:
   - A workspace that allows students to create and edit solutions in flowchart form.
   - A compile module that checks program logic and expressions.
   - A debug window that shows the values of program variables during program running, and
   - An updatable databank of problems for students to solve;

2. Create the system using C Sharp (#), Extensible Markup Language (XML), and Cascaded Style Sheets (CSS);

3. Test and improve the system in terms of functionality, and reliability; and

4. Evaluate the performance of the developed system using ISO 9126 criteria for quality software.

Conceptual Model of the Study

The Conceptual Model of the Study, as shown in Figure 1, explains the essential components in the development of the system. The components include: input, process, and output. The study requires basic understanding of a typical structure of an IDE, Program Logic Formulation, Flowchart, Database, Compiler Theory, Finite Automata, C#, CSS, and XML. The Software requirements include a 32 or 64-bit Windows Operating System, and Visual Studio. The process specifies the steps in developing the system. System design pertains to the general feature and function of the system using the Finite Automata, Entity Relationship Diagram, and System Flowchart. The system development incorporates the result of system analysis to code the requirements of the study which includes program coding. Finally, the study was subjected to system testing and debugging to find errors and to attain perfection based on the set design characteristics. The output of the system is "An IDE in Program Logic Formulation" which was evaluated by purposively sampled respondents to assess its performance.

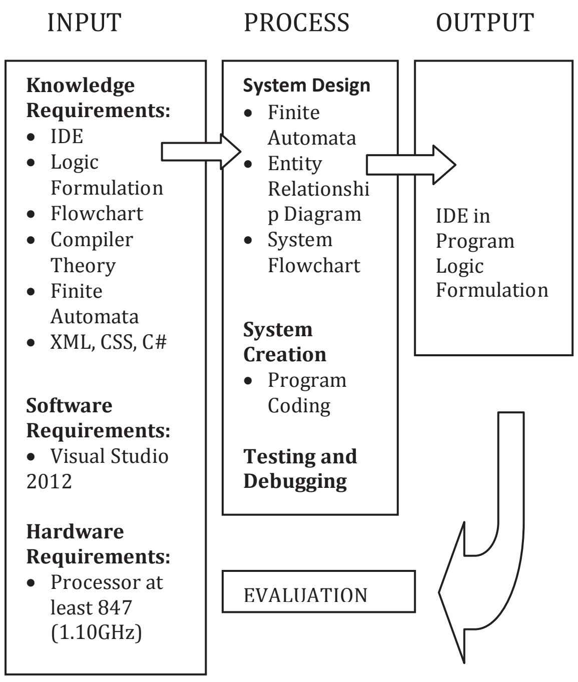

Figure 1. The Conceptual Model of the Study

Literature Review

Integrated Development Environment

O'Dell (2010) defined integrated development environment (IDE) as a software application that gives complete facilities to computer programmers for program development. The IDE performs as text editor, debugger, and compiler all in one sometimes stuffed, but generally a useful package. Some IDEs contain compiler, interpreter or both such as Microsoft Visual Studio and Eclipse; others do not such as Sharp Develop and Lazarus. The objective of the IDE is to lessen the configuration required to piece together various development utilities, instead of providing the same set of capabilities as a unified unit. Reducing the setup time can increase developer productivity. For instance, code can be constantly parsed, while it is being edited, providing instant feedback when syntax errors are detected. That can speed the program development and analysis of the logic.

The project study developed by Mendoza (2006), an IDE for Turbo Assembler, provides an environment where primary development tools such as an editor, assembler, linker, and debugger are loosely tied in the package not requiring the users to run each application in the command line. This software assisted novice assembly language users in running assembly programs and locating errors. Its text editor is line numbered to easily identify the corresponding error messages produced. However, these messages are owned error messages of the Assembler in contrast to the error messages used by the researcher of this study. The error messages are simple, clear, and specific. The researcher designed the IDE's own analyzer that reads, screens, and analyzes correct format for input and output operations as well as the expressions such as mathematical, and conditional. The developed IDE used and invoked methods under C#'s Compiler class similar to IDE for Turbo Assembler.

Logic Formulation

Farrel (2002) stated that once the programmer has developed the logic of a program, then and only then can he or she write the program in one of many programming languages that exist. This point the programmer can start worrying about the proper format in using each command in the program. The program's output is wrong when it contains logical errors, however, a program with syntax errors cannot execute.

Stated in the paper of Stachel (2013), research on techniques for teaching computer programming to novice learners has suggested that introducing programming concepts and theories is extremely difficult, because the learners have to adapt syntactical methods of the programming language as well as their interface to the programming world. Computer programming for the novice requires the understanding of a variety of different areas such as logic and mathematical concepts, syntax, the language interface, algorithms, flowcharts, and *pseudocode* associated with programming theory.

The heart of programming lies in planning the program's logic during this stage of the programming process; the programmer outlines the steps, determining what steps to take in and how to order them in the program. Solutions to problems can be done in many ways. Flowcharts and *pseudocode* (pronounced "sue-dough-code") are the two most common tools (Farrel, 2002).

Gill (2004) emphasized the benefits of using the flowcharts as a teaching tool in introductory programming classes. Flowcharting has been an important component of the overall learning in the course. In addition, analysis of survey data gathered from students suggested that learning flowcharting early in the course has benefited their learning in subsequent programming assignments.

Commercially available system produced by Matrix Multimedia, Flowcode is a flow chart programming language. This makes flowcode an excellent introduction into programming programmable interface controllers (PICs). Its flowchart programming method allows users with little experience to create an electronic system without writing traditional code line by line. There are 3 steps to program PICs: 1) Design using the drag and drop flow chart icons and electronic components on screen; 2) Simulate the designed electronic system, program. The menu controls allow users to step through each icon in the program and show its effects on the screen. Test the system's functionality if it still manages to produce same output; and 3) lastly, download the design. Flowcode sends the program to the PIC micro microcontroller device. Behind the scenes the flowchart is turned into C-code which is then compiled by *Boost C compiler*.

Memory Variable and Naming Convention

Data are entered into the program using any input devices and the data entered for processing are saved on a certain location. This location is called variable (De la Rosa, 2008).

Variables like humans need names to be identified. Each programming language has its own error message, syntax error, for not following the rule for variable naming. The language interpreter uses a kind of system finite automata to model the correct structure for variable names.

Finite Automata

Commonly used programs such as text editors and lexical analyzers found in most compilers are often designed as finite state systems. For example, a lexical analyzer scans the symbol of a computer program to locate the strings of characters corresponding to identifiers, numerical constants, reserved words, and so on. In this process the lexical analyzer needs to remember only a finite amount of information, such as how long a prefix of a reserved word it has seen since startup. The theory of finite automata is used heavily in the design of efficient string processors of these and other sorts.

Martin (1997) defined a Finite Automaton, or finite-state machine (abbreviated FA) as a 5-tuple $(Q, \Sigma, q0, A)$, where Q is a finite set (whose elements are states) Σ is a finite alphabet of input symbols $q0 \in Q$ (the initial state) $A \subseteq Q$ (the set of accepting states) δ is a function from $Q \times \Sigma$ to Q (the transition function)

For any element q of Q and any symbol $\alpha \in \Sigma$, then $\delta(q, \alpha)$ as the state to which the FA moves, if it is in state q and receives the input α.

Likewise, *Hopcroft* and *Ullm*an (1979) described finite automaton. (FA) as consisting of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from an alphabet Σ. For each input symbol there is exactly one transition out of each state (possibly back to the state itself). One state, usually denoted q0, is the initial state, in which the automaton starts. Some states are designated as final or accepting states. A directed graph, called a transition diagram, is associated with an FA as follows. The vertices of the graph correspond to the states of the FA. If there is a transition from state q to state p on input *a*, then there is an arc labeled *a*from state q to state p in the transition diagram. The FA accepts a string *x* if the sequence of transitions corresponding to the symbols of *x* leads from the state to an accepting state. The FA plays role in computer design and parsing, the typed programs are broken into tokens and then those tokens are recognized through Automata theory which happens in the first state of the compiler design.

Compiler Theory

According to *Muhammad* (2013), the compiler is a program that reads a program in one language, the source language and translates into an equivalent program in another language. There are two parts of compilation – the analysis phase and synthesis phase. In the former the syntactic structure and some of the semantic properties of the source program are computed. The computed properties are called the static semantics. This includes all semantic information that can be determined only from the program without executing it with the input data. The results of analysis phase consist of either messages about syntax or semantic errors in the program or a suitable representation of the syntactic structure and static semantic properties of the program. This phase is independent of the properties of the target language and the target machine. The synthesis phase constructs the desired target program from the intermediate representation.

For his part *Gill* (2004) developed a software called Flow C. It allows the user to view the code implied by each symbol drawn in the flowchart, however, it requires the flowchart to be syntactically correct. This means the user drawn flowcharts are valid in terms of input and output, mathematical, logical, and relational expressions. These are different from the researcher's study. The designed IDE displays error messages whenever parsed invalid expressions are encountered. Also, Flow C was designed to generate complete applications that may then be compiled and run in MS Visual Studio.NET which is an example of object-oriented programming language.

C #

C# was modeled after C++ programming but some difficult features to understand in C++ have been eliminated in C#. It is very similar to Java because Java was also based on C++. In Java, simple data types are not objects unlike in C# every piece of data is an object, providing all data with the functionality of true objects. C# contains a GUI interface that makes it similar to Visual Basic but considered more concise than Visual Basic.

Abella, et al. (2013), developed a software using the language C#. Their study was designed to translate the flowcharts into their equivalent C language in proper syntax. The GUI aspect of C# was used to design the anticipated appearance of the software. Also, the conversion from user-provided in flowchart to C# was covered through accessing the external file by C# compiler. Similarly, with the developed IDE, it accessed the Compiler parameters of C# so that each output of the symbol on the console command will be read and passed back to the IDE itself. However, the flowchart symbols of the developed IDE are Extensible Markup Language format, while the first one is nodes constructed using C# controls such as *label* and textbox. Labels, textboxes, and buttons are classes that can be instantiated as object in object oriented programming.

Extensible Markup Language

Extensible Markup Language is a perfectly good vehicle for describing data to be transmitted over the Internet. XML is not used for describing the semantics of data neither replaces data modeling and database design (Hay, 2007).

According to Ferrara of webdesign.com, the key to viewing XML in a browser is Cascading Style Sheets. Style Sheets allow designers more flexibility when creating the look for a web page and define every aspect of an XML document, from the size and color of the text to the background and position of the non-text objects. Cascading Style Sheet or CSS allows the Web designer to control the page layout or a simple way to add style such as font, colors, spacing, and so forth. HTML deals with structure, while CSS deals with style. CSS and HTML are having some noticeable similar attributes in names and values.

ISO 9126 Software Quality Model

Table 1, ISO 9126 model, presents the criteria with corresponding attributes that indicate quality software.

Table 1
The ISO 9126 Model

Criteria	Attributes
Functionality	The presence of set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
Reliability	The capability of the software to maintain its functionalities under specified conditions for specified period of time.
Usability	The usefulness of the software to meet the user needs, and on the individual assessment of such use, by a stated or implied set of users.
Efficiency	The level of performance of the software and the amount of resource used to meet the required functionalities.
Maintainability	The effort needed to make specified modifications in the software.
Portability	A set of attributes that bear on the ability of software to be transferred from one environment to another.

Method

Description of the Respondents

The respondents composed of 20 faculty members handling computer course, 10 students from BSIT, BSCS, and BSIS courses, and 5 technical experts from the College of Science were selected using purposive sampling technique.

Research Design

The study used the developmental method of research. The developed IDE is designed to assist students in learning programming through flowchart. Its interface is most likely similar to other existing IDEs including the window naming such as workspace, toolbox/object controls, and message. Its interface design introduces students to Object Oriented Programming (OOP).

Research Instrument

The IDE was evaluated by 35 respondents using ISO 9126 criteria for quality software. Specifically, the IDE's functionality and reliability were tested by drawing different sets of solutions.

The respondents rated the software using a 5-point Likert scale with 5 being the highest and 1 being the lowest.. The data were collected and the overall mean was computed, the results were interpreted using the scale presented range in Table 2 and the corresponding qualitative interpretation.

Table 2
The Scale Range and its Qualitative Interpretation

Range	Qualitative Interpretation
4.51 – 5.00	Excellent
3.51 – 4.50	Very Good
2.51 – 3.50	Good
1.51 – 2.50	Fair
1.00 – 1.50	Poor

Results

The main screen of the IDE has five (5) major parts: (a) Command window, (b) Message window, (c) Problem window, (d) Flowchart Controls Window, and (e) the Workspace, as shown in Figure 2.

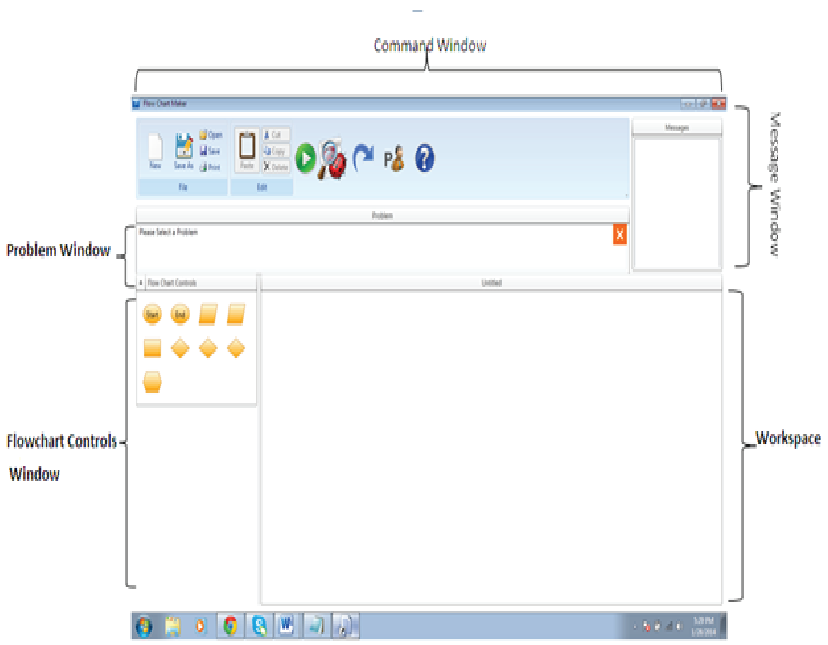

Figure2. Main Screen and the Parts of the Developed IDE

Table 3
Flowchart Maker and Problem Manager Test Results

Module	Steps Undertaken	Observation/Result
Flowchart Maker	1. Tested to click and drag each flowchart symbol to the workspace. 2. Tested to connect one symbol to another symbol. 3. Tested to copy and delete flowchart symbols within the workspace. 4. Tested to create and save flowchart file. 5. Tested to load existing flowchart file	1. The flowchart symbols can be clicked and dragged. 2. They can be connected to other symbols within the workspace. 3. They can be copied and deleted from one workspace to another or within the workspace. 4. The IDE can create new and save flowchart file. 5. The IDE can load existing flowchart files.
Problem Manager	1. Tested to add problems in the database. 2. Tested to add test data for each problem in the database. 3. Tested to retrieve problems from the database.	1. Problems can be added in the database. 2. Test data were tagged to the selected problem. 3. Problems can be retrieved.

Test Results

The following tables present the results of the test conducted.

Table 3 shows the test results for flowchart maker and problem manager modules. The IDE can create, edit, open, print, and save flowchart. The symbols can be copied, deleted, and connected to other symbols within the same workspace. Similarly, the problem manager can add, update, and retrieve problems and test data.

Table 4
Compile Module Test Results

Steps Undertaken	Observation/Result
A. The following steps tested the Compile module using a problem retrieved from Choose Problem module.	
1. Tested to retrieve problem: Insect Population.	1. The Insect Population problem was retrieved.
2. Tested to click Compile icon for the following flowchart with: • Uninitialized variable Initial, and typed input on the output symbol. • Wrong formula: GrowthRate=(Pop WekLater-Initial Pop)/ PopWekLater • Correct logic and inputs: 100 and 130, and 80 and 120.	2. Compile icon can be clicked and; • Can check correct syntax for variable declaration and output operation, see Appendix A Figure 14. • Can check wrong solution or incorrect logic based on Test data, see Appendix A Figure 15. • Can check correct logic based on Test data, see Appendix A Figure 16.
3. Observed the message for correct/incorrect logic and error messages.	3. The Compile module can check the program expressions and operations.
B. The following steps tested the compile module using not a problem-based flowchart.	
1. Tested to draw flowchart that computes the area of a triangle (T) or square (S).	1. The flowchart can be drawn.
2. Tested to click Compile icon for the following flowchart: • Used a word equal instead of (=) operator inside the selection symbol: figure equal "s". • Unbalanced the parentheses in the arithmetic expression: Area= (b*h/2. • Used valid expressions for all symbols.	2. Compile icon can be clicked and; • Can check correct syntax for conditional expression, see Appendix A Figure 17. • Can check correct syntax for arithmetic expressions, see Appendix A Figure 18. • The Compile module can check program expressions and operations, see Appendix A Figure 19.
3. Observed the message for error messages and syntactically correct flowchart structure.	3. The compile module can check the correct program expression and operation.
C. The following steps tested the compile module if it checks correct flowchart structure: one Start symbol, more than two End symbols, and no connections on symbols.	
1. Tested to create a flowchart with more than one End symbol.	1. The Compile module accepted the flowchart structure. This means that the compile module can check PLF rule, see Appendix A Figure 20.
2. Tested to create a flowchart with more than Start symbol.	2. The Compile module produced an error message: *"Error: Multiple Start"*, see Appendix A Figure 21.
3. Tested to create a flowchart with symbols not connected and no label path for decision symbol.	3. The Compile module produced an error message: *"Error: Lack of Connections and Error: Invalid Label, T/F only"*, see Appendix A Figure 22.

Table 4 shows the testing done in the IDE's compile module. There are three scenarios, *A*, *B*, and *C,* were used to test the IDE. First scenario, it checks the logic of the flowchart, that is a solution to the problem retrieved from Choose problem module, it produces messages pertaining to each given. The IDE detects undeclared variable, wrong solution/ logic, and correct solution/logic.

Table 5
Debug Module Test Results

Steps Undertaken	Observations/ Results
The following steps tested the Debug module using flowchart that displays where the axis or quadrant of a point lies.	
1. Tested to click the Debug icon.	1. The Debug icon could be clicked then it showed the Debug Program window.
2. Tested to simulate using another input:-3 and 5.	2. The Debug Program window displayed variables values and the expected output on the Output window: *"2"*, see Appendix A, Figure 23.
3. Tested to simulate using input: 2 and 2.	3. It displayed variables values and the output: *"1"*, see Appendix A, Figure 24.
4. Tested to simulate using input: -10 and -20.	4. It displayed variables values and the output: *"3"*, see Appendix A, Figure 25.
5. Tested to simulate using input: 0 and 0.	5. It displayed variables values and the output: *"origin"*, see appendix A, Figure 26.
6. Observed the simulation.	6. The Debug module can simulate the flowchart.
The following steps tested the Debug module using flowchart that obtains the possible roots of the linear equation: $4x + 3y - 9z = 5$ from 0 to 5.	
1. Tested to click the Debug icon.	1. The clicked Debug icon showed the Debug Program window.
2. Observed the simulation.	2. The Debug module displayed 2, 2, and 1. Then 2, 5, and 2. Then 5, 1, and 2. Then 5, 4, and 3, see Appendix A, Figure 27.

Table 5 shows the test results done in the IDE's debug module. The flowchart that displays the axis/quadrant where a point lies is debugged using 4 sets of inputs and the results are noted. First test input is (-3, 5) then the output is *2,* meaning the point lies in quadrant two. Second, test input is (2, 2) then the output is quadrant *1*. For test input (-10,-20), debug window displayed the output *3*.

The flowchart that determines all possible roots of the linear equation, $4x + 3x - 9z = 5$, is shown in Figure 27.

Table 6
Reliability Test Results

Steps Undertaken	Observations/Results
Problem A. *Division by Zero*. Write a flowchart that accepts and divides two integers.	
1. Tested to solve a problem that performs the division of two integers.	1. The IDE can draw the flowchart.
2. Tested to input two integers for variables x and y, 5 and 0, respectively.	2. It can store the values, 5 and 0, into variables, x and y, respectively.
3. Observed the simulation.	3. It can continue execute division by zero instruction. Showed on the Output window is *Infinity* value for variable z. See Appendix B, Figure 28.
Problem B. *Nonterminating Loop*, Write a flowchart that displays the numbers from 0 to 10.	
1. Tested to draw the flowchart using Post- test loop.	1. The IDE can draw the flowchart.
2. Tested to set the conditional expression: *ctr> 0*. Observed the simulation.	2. The IDE can manage to continue execute in a non-terminating environment. See Appendix B, Figure 29.

Problem C. *Not A Number (NAN) result on Modulo operation*. Write a flowchart that scans pairs of integers until it reaches a pair in which the first integer evenly divides the second.	
Using Post-test Loop and Pre-test Loop	
1. Tested to draw flowchart that scans two integers continuously.	1. The IDE can draw the flowchart.
2. Tested to input set of integers: 4 and 21, 3 and 16, 0 and 10, and 5 and 25.	2. It can accept integers.
3. Observed the simulation.	3. It can manage to continue execute instruction 10 Mod 0. The output showed is *25 and 5.* See Appendix B, Figure 30 and Figure 31.
Problem D. Write a program that determines where the quadrant or axis of a point, x and y coordinates, lies	
Multiple Selection Structure and Nested Multiple Selection Structure	
1. Tested to draw the flowchart.	1. The IDE can draw flowchart.
2. Tested to input point 5 and 0.	2. It can accept input: 5 and 0.
3. Observed the simulation.	3. It showed the output: "x-axis". See Appendix B, Figure 32 and Figure 33.
Problem E. Write a program that accepts an integer. The program outputs the sum of its digits. For example 564, the output is 15.	
Pre-test Loop and Post-test Loop	
1. Tested to draw flowchart.	1. The IDE can draw flowchart.
2. Tested to input an integer, 564.	2. It can accept input: *564*.
3. Observed the simulation.	3. It showed the output, *15*. See Appendix B, Figure 34 and Figure 35.

Table 6 shows the reliability test results of five problems, Problem A to E solved using different solutions. Problem A is a flowchart that tries to divide *5* by 0; it gives a *NaN* result, instead of force exit from program which happens for some structured programming language. Flowchart for Problem B has a non terminating loop, however, the IDE manages to display the output and continues the program. The test for Problem C is similar to Problem except that it a *mod* operator in which the second operand is *0*. Problem D uses two strategies, multiple-selection and nested multiple-selection structures, both solve it correctly and produce the same output. Problem E is solved using two different solutions: post-test and pre-test loops, but shows the same output.

The IDE and Flowcode

The Flowcode and IDE for PLF are both designed to assist users with little experience in programming. Flowcode helps PIC programmers to design, simulate, and test the functionality of the electronic systems before sending the program to microcontroller device in minutes, while the latter helps novice programmers to formulate correct steps to solve a given task before translating to computer language. Both require its users to have a correct logic.

Flowcode and IDE both have GUIs containing the standard flowchart symbols such as input/output, process, decision, and terminal. Symbols are drag and drop icons with each icon equivalent to one line of program code. Both have controls within the environment allowing users to start, stop, pause, and step through their program one icon at a time.

Evaluation Results

The developed system entitled "An Integrated Development Environment in Program Logic Formulation" gained an overall mean of ***4.62*** with a descriptive rating of **Excellent**.

Table 7
Summary of Respondents' Rating (n=35)

Criteria	Mean	Descriptive Rating
Functionality	4.68	Excellent
Reliability	4.53	Excellent
Usability	4.72	Excellent
Efficiency	4.54	Excellent
Maintainability	4.60	Excellent
Portability	4.66	Excellent
Overall Mean	4.62	Excellent

Table 7 shows the summary of mean rating per criterion. It further yields that the highest mean score is 4.72 obtained by the criterion, *Usability*. The criterion *functionality* obtained an overall mean of 4.68 with a descriptive rating of excellent. *Efficiency* got a mean of 4.54. The indicators of *maintainability* garnered a mean of 4.60, interpreted as excellent. *Portability* obtained a mean of 4.66 interpreted as excellent, while *Reliability* a mean of 4.53 interpreted as excellent.

Discussion

Based on the results of the test and evaluation conducted, the developed IDE can provide a working environment where students can create, save, open, edit, and print flowcharts. It can check invalid expressions for input, output, mathematical, relational and logical and display the error for invalid expressions. Also, it can simulate flowchart from Start through End symbols detect the presence of wrong logic in the flowchart, capable of maintaining storage of questions and its test data.

The following are the summary of evaluation:

1. **Functionality**. The evaluators rated the system as excellent. This means that the software performs the task required accurately, its components interact properly with each other, complies with the needs of the students, and is protected from unauthorized access.

2. **Reliability**. The evaluators rated the system as excellent which proves that the software failures occur less often, has resistance to failures, and has the ability to recover itself after failure.

3. **Usability**. The evaluators rated the system as excellent. It obtained the highest mean rating of *4.72*. This could be attributed to the evaluators' comments that the software is easy to learn, understand and operate since the software runs on a widely used Windows environment.

4. **Efficiency**. The evaluators rated the system as excellent. The software responds quickly and utilizes resources efficiently. The mean rating falls within the range of the scale value excellent, which indicates that all the indicators have been satisfied.

5. **Maintainability**. The evaluators rated the system as excellent, its indicators of *maintainability*: software failures are easy to diagnose, the software continues to operate even if changes had been made, and the software is easy to test.

6. **Portability**. The evaluators rated the system as excellent. This means that the IDE can be easily installed, its components configured, and it complied with portability standards.

Conclusion

In light of the aforementioned findings, the following conclusions are drawn:

1. The IDE for Logic Formulation was successfully designed such that:

   - Logic written in flowchart form can be created, saved, modified, and printed.

   - Logic can be checked and verified based on the required output stated in the problem. Correct format for input and output operations, mathematical, relational and logical expressions are verified.

   - Logic can be visually followed through a Debug window. Variables and its contents are displayed on the Debug window to understand logic flow.

2. The IDE was successfully created using C#, CSS, and XML.

3. Tested according to functionality and reliability, the system found that all modules performed the tasks as designed.

4. The developed system was evaluated as Excellent in terms of functionality, reliability, usability, efficiency, maintainability, and portability, which proves that the system can be a useful tool to develop students' skills in programming.

Recommendations

The following are suggested for further enhancement of the developed software.

- Translator from flowchart to C codes may be included to achieve better performance of the students in the subject Computer Programming

- Included additional data type such as Array may widen the scope for possible answers.

- Modular programming be made a part to make the software more suitable for Computer Programming 1 syllabus.

- On-page Connector be included to present a more organized flow lines within the flowchart.

References

Book

De la Rosa, Jerald H. (2008). *Simple: Program Logic Formulation*. Philippines: Andes Mountain Printers

Farrell, Joyce. 2002. *Programming Logic and Design*. Canada: Course Technology Thomson Learning.

Hopcroft, John E. (1979). *Introductionto Automata Theory, Languages and Computation*. Canada: Addison-Wesley Publishing Company, Inc.

Martin, John C. (1997). *Introduction to Languages and The Theory of Computation*. Singapore: The McGraw-Hill Companies, Inc.

Van Dam, Bert. (1988). *Flowcode 6*. Netherlands: Wilco, Amersfoot © Elektor International Media 2014.

Unpublished Thesis

Abella, Rubeltio R., et. al. (2013). Creating a Data Flow Paradigm Programming Language Written in C#. BSCS Thesis. Ama Computer College.

Mendoza, Benjami., et al. (2006). "Integrated Development nvironment for Turbo Assembler." BSC major CS Thesis. Technological University of the Philippines,

Journals and Other Publications

Gill, T. G. (2004). Teaching Flowcharting with *Flow C. Journal of Information Systems*. Education, *Vol 15* Issue 1: 65 – 77.

Hay, David C. (2007). Data Structure: Data Modelingor XML. *The Data Adminstration News Letter*, 28-30.

Stachel, J., Marghitu, D., Brahim, T. (2013). Managing Cognitive Load in Introductory Programming Courses: A Cognitive Aware Scaffolding Tool.

Journal of Integrated Design and Process Science, *Vol. 17* Issue 1:37-54.

Electronic Sources

Ferrari, Darla.(2014). XML and CSS. Retrieved from Web design website: http://www.webdesign.com/odbeginner/xml/a/xml-and-css.htm.

Muhammad, Rashid Bin. (2013). *Compiler Lecture Notes*. http://bscs16.blogspot.com/2011/04/dc-lecture-notes_06.html.

O'Dell, Jolie. (2010). *A Beginner's Guide to Integrated Development Environment*. Retrieved from Mashable website: http://mashable.com/2010/10/06/ide-guide.