ISSN 1533-9211

DOI 10.5281/zenodo.8241305

ONLINE PSEUDOCODE INTERPRETER

LORETO G. GABAWA, JR.[1]*, CHRISTIAN LESTER D. GIMENO[2] and Dr. TRACY N. TACUBAN[3]

[1,2,3]Iloilo Science and Technology University, Burgos St. Lapaz Iloilo City, Philippines.
Email: [1]loreto.gabawa@isatu.edu.ph (*Corresponding Author), [2]christianlester.gimeno@isatu.edu.ph, [3]tracy.tacuban@isatu.edu.ph

Abstract

First-year students at Iloilo Science and Technology University, located in Iloilo City, Philippines, should enroll in the Programming Logic Formulation course before their actual programming classes. This course must introduce them to programming concepts since most students do not have programming classes in their junior and senior high schools. In this course, students write pseudocodes to represent their solutions for a programming problem, and these are written on paper and submitted to the instructors for checking. The researchers conducted this study to create an online Integrated Development Environment (IDE) for Programming Logic Formulation, specifically in designing, writing, and checking pseudocode. The study includes software design that can perform lexical analysis, syntax analysis, and parsing of pseudocode using a recursive descent parsing algorithm. The software was designed as an online platform and can accept sequential, selection, iteration, and recursive functions and desk check tables. The system was evaluated based on International Organization for Standardization (ISO) 9126, and the evaluation result was described as "Excellent." The positive evaluation results reinforce the system's potential as a practical educational resource, enhancing students' programming readiness and empowering first-year students to develop a solid foundation.

Keywords: Integrated Development Environment, Pseudocoding, Programming Logic Formulation, Parser

INTRODUCTION

As stated in the Official Gazette of the Philippine government, secondary schools offer a two-year specialized upper secondary education for Senior High School students. The Core Curriculum and Specialized Tracks offer subjects for this curriculum. There are seven (7) Core Subjects: Social Sciences, Natural Sciences, Languages, Philosophy, Mathematics, Literature, and Communication. The specialized track has the following tracks: Technical-Vocational-Livelihood, and Sports and Arts. The academic track includes Humanities, Science, Technology, Engineering, and Mathematics (STEM), Education, Social Sciences (HESS), Accountancy, Management (BAM) and Business (Official Gazette of the Republic of the Philippines, n.d). The Technology and Livelihood Education (TLE) and Technical-Vocational-Livelihood (TVL) Track has the following strands: Industrial Arts, TVL, Maritime, Agri-Fishery Arts, Information and Communication Technology, and Home Economics (Department of Education, n.d).

First-year students applying to the University came from either one of these strands. Acknowledging this situation, the University offered a bridging course for first-year students to introduce them to the programming concepts vital in their programming classes. Such innovation is a way for the University to equip first-year students with the necessary background and skills in computer programming, especially for those from the Academic,

ISSN 1533-9211

DOI 10.5281/zenodo.8241305

Sports, and Arts strands. Jones (2019) conducted a study stating that the attrition rate of computer programming students also contributes to students' difficulty in their computer programming classes. According to Jones, it would be easier for students to grasp the higher-level topics of the course when they need to learn the necessary programming concepts at the early stage of the course.

Moreover, according to Ramat et al. (2011), since programming concepts are interdependent, students must learn basic skills before advancing to higher courses. Students must understand the basic structure of the program, the proper use of syntax and semantics, and how to translate algorithms into program code which should also follow the appropriate programming syntax. Furthermore, according to Hsiao (2022), developing the students' programming logic ability is essential before their actual programming courses. Students who are new to these concepts will find classes in computer programming challenging and may soon lose interest in studying the course. The authors suggested strategies to motivate and help students understand programming concepts. First, the teachers must teach the students how to write their self-designed programming codes. Secondly, the teachers should provide the students should with a user-friendly interface where they can write the program codes. The last step includes enhancing the students' problem-solving abilities.

With this literature, the campuses of the University located at La Paz, Miagao, Dumangas, and Barotac offer the Information and Communication Technology (ICT) programs such as Bachelor of Science in Information Technology, Bachelor of Science in Computer Science, and Bachelor of Science in Information Systems has integrated the course Program Logic Formulation in its curriculum. The course introduces the fundamental concept of programming to students and the use of control structures in the program. Teachers teach the students how to write their algorithm and pseudocode to represent their solution to a programming problem, use an IPO chart, conduct test checking to allow them to test and evaluate the correctness of their answers, and draw flowcharts to represent their algorithm. According to Farell (2015), an algorithm is a set of detailed, clear, and ordered steps or rules in solving a problem. On the other hand, a pseudocode represents an algorithm in a computer programming-like structure, but it does not follow any programming syntax and rules. Desk Checking is a process of checking and testing the correctness of an algorithm. These are the concepts that the students are learning in the Program Logic Formulation class.

However, teaching students how to solve programming problems using pseudocodes is also challenging to students and educators. Faculty members have adopted a traditional way of teaching programming fundamentals and found it very difficult because the students wrote the algorithm and pseudocode n paper, and manually checking the paper requires time and effort for the faculty handling the course. On the other hand, it is also difficult for students since they are not provided with a way to code and check their pseudocode in an Integrated Development Environment (IDE) because there is no available IDE for pseudo coding in the University.

Following the strategies from Hsiao, the researchers developed a tool for learning logic programming fundamentals so that students with no experience in computer programming could use them. Using the tool, the students can understand the basic concepts and principles

The Seybold REPORT

ISSN 1533-9211

DOI 10.5281/zenodo.8241305

of algorithms effectively because it provides visualization and automatic interpretations, as suggested by Hsiao. The researchers developed this study to give students and faculty members handling the Programming Logic Formulation course additional materials on how to write, check and evaluate the correctness of their algorithm and pseudocodes.

The software's design employs lexers to split pseudocodes and convert them into tokens. The abstract syntax tree uses a parser designed using the descent parsing algorithm to process the tokens as input. This process will determine whether the pseudocode is designed correctly according to a predefined format. The software also can support recursive functions, arrays, loops, if statements, and other control structures. In addition, the newly developed software is a web-based pseudocode interpreter, allowing users to use the platform on their own time and at any location, such as their homes. Users can also use the platform through mobile devices such as smartphones since it is a mobile-responsive application.

OBJECTIVES OF THE STUDY

The study includes developing and evaluating an online Pseudocode Interpreter that should be able to read and perform recursive functions, including a Desk Check Table for the interpreted pseudocodes. The development system is evaluated based on the eight (8) software characteristics of the International Organization of Standardization (ISO) 9126

MATERIALS AND METHODS

Pseudocode Interpreter software was developed and tested as part of Software Development research to determine its conformity based on the ISO 9126 standards. The study is also Descriptive Research. According to Rillo and Alietio (2018), descriptive research is a method used to gather, analyze and tabulate data for interpretation. Descriptive research can include calculating the measure of central tendency and data variability. In this study, the researchers interpreted the result of the evaluation using the average mean and standard deviation to represent the perception of the evaluators of the survey as to the effectiveness of the software.

The researchers also used the prototyping model to develop the software. Software prototyping, according to Dennis et al. (2012), allows researchers to create the working model of the system. The researchers then presented this model to the client and introduced further enhancements to the software based on the client's suggestions. This cycle continues until the software meets all the requirements and the client is satisfied with the product. Like other process models and software development life cycles, prototyping methodology started with essential requirement identification. In this phase, the researchers identify the functional and non-functional requirements of the system. It includes the software's features, user interface design, and other related user requirements. The prototype development phase follows the process. In this phase, the researchers developed the initial software prototype. During this phase, the researchers developed the diagrams needed by the programmer during the coding stage. The prototype is a work in progress. The enhancement of the system is continuous based on user suggestions. The researchers also conducted requirement analysis for developing the prototype using a Data Flow Diagram (DFD) that illustrated how the data is processed and how the software accepts

ISSN 1533-9211
DOI 10.5281/zenodo.8241305

inputs and generates results. After the development of the prototype, it underwent the evaluation process. The researchers presented the software prototype to the end users to gather feedback and improve the system's design and functionality. After each modification, the software is presented again to the client and the entire process is done repeatedly until the client is satisfied.

Finally, the researchers proceeded to revisions and enhancements of the prototype. The system was initially written using Java and Java Server Pages and is improved using the Javascript platform. The researchers redesigned the system to accept inputs for the iteration structure, which was not present in the previous design. The researchers also incorporated the gathered feedback from the end users on the design prototype in developing the final software. After the construction of the final software in the implementation phase, the system undergoes evaluation based on ISO 9126 software quality standards.

Every programming language has a unique grammar. The researchers used the Extended Backus-Naur Form (EBNF) in this study. An EBNF, as stated in harvard.edu (2006), is a grammar that accepts several lines of codes, and each line includes a set of symbols, a pattern that may be used instead of the symbol and a colon. The nonterminal symbol of the EBNF is placed on the left-hand side of the grammar. Some symbols that the EBNF can read include variable names, integers, strings and literals. Below is the EBNF Grammar for the pseudocode created by the researchers in this study.

```
   <pseudocode>                ::= “BEGIN” <block> “END”
   <block>            ::= <decl-part><stmt-part>
   <decl-part>                 ::= [<variable-declaration-part>]
   < variable-declaration -part>          ::= < variable-declaration > | <array-variable-declaration>
              { <var-decl> | <array-var-decl> }
   < variable-declaration >          ::= <identifier>
   <identifier>                ::= <LowerCase >|<UpperCase>
             ”_”< LowerCase >|< UpperCase >|<integer>
             { < LowerCase >|< UpperCase >
             ”_”< LowerCase >|< UpperCase >|<integer> }
   < LowerCase >    ::= ‘a’ | ‘b’ | ‘c’ | ‘…’ | ‘z’
   < UpperCase >    ::= ‘A’ | ‘B’ | ‘C’ | ‘…’ | ‘Z’
   <stmt-part>                 ::= { <stmt-sequence> | <stmt-condition>
|<stmt-repetition> }
   <stmt-sequence> ::= <assignment> { <assignment> }
   <assignment>                ::= <identifier> “=” <expression>
   <expression>                ::= <boolean-expression>
   <BooleanExpression>::= <BooleanTerm> [<or-op>< BooleanTerm >]*
   < BooleanTerm >             ::= <NotFactor> [AND <NotFactor>]*
   <NotFactor>                 ::= [NOT] <BooleanFactor>
   <BooleanFactor>                ::= <BooleanLiteral> | <BooleanVariable> | <Relation>
   <Relation>                        ::=         <ArithmeticExpression>         [<RelationOperator>
          <ArithmeticExpression>]
   <ArithmeticExpression>   ::= <Term> [<add-op><Term>]*
   <Term>                            ::= <SignedFactor> [<Mul-op><Factor>]*
   <SignedFactor>              ::= [<minus-op>] <Factor>
   <Factor>                          := <Integer> | <Variable> | (<Expression>)
   <integer>                         ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
   <boolean-value>             ::= true | false
```

ISSN 1533-9211

DOI 10.5281/zenodo.8241305

The software takes the various symbols or characters as lexemes through the lexer. A lexer, according to Peisert (2002), is a code that can analyze the lexical structure of the codes. The lexer will read the characters entered by the users, in this case, the various pseudocode, take each character one at a time and return each token after the lexer has read it. These tokens, called lexemes, will then be matched to a predefined pattern for NUMBER, STRING, KEYWORD, and IDENTIFIER, among others. After which, the parser will then analyze the token produced by the lexer. The parser will extract meaningful structures like functions, classes, variables, expressions, and statements from the pseudo-code. The parser will then place each token in the Abstract Syntax Tree and check if the pseudocode is written according to the grammar of the pseudocode interpreter. If the parser finds any syntax errors, it will generate the appropriate error messages, and if the pseudocode is correct, it will display the corresponding message.

Like many versions of pseudocodes at present, the version below has certain conventions adopted in this study:

1. Pseudocode to be easily understood are written in English form.
2. Statements can be defined as output and input.
3. Every statement is written one after the other. One line of code is written on top of another for readability.

The interpreter uses several predefined statements based on existing rules written by Robertson (2006).

A. General Rules

1. A pseudocode must have a BEGIN keyword at the beginning and an END keyword at the end.
2. To signify control structures, the user must use keywords and indentions.
3. Every statement or line of code is written one after the other
4. Follow the variable naming convention
   a. A variable name must begin with a letter followed by a combination of the letters and numbers.
   b. If a variable name is composed of more than one word, use an underscore to join those words.

      Examples: first_name, first_number

      Examples: firstName, firstNumber, firstSecondNumber
5. A variable name must not have a space
6. The supported data types are Integer and String
7. A variable declaration does not precede a data type but must contain an assignment of

ISSN 1533-9211

DOI 10.5281/zenodo.8241305

value.

Examples:

student_count = 21	- variable with an integer value
message = “hello world”	- variable with string value

a) The Output Statement

Output statement uses the keywords such as PRINT, DISPLAY, OUTPUT and PRINTLN as predefined functions to display specific text or strings, variable values and constants.

A.1. PRINT syntax with examples:

```
PRINT “Hello world”
OUTPUT number
DISPLAY 1234
PRINT num1 + num2
PRINT num + 12
```

A.2. PRINTLN keyword or function syntax with examples:

The PRINTLN keyword or function is used to display text or string, variable value or constant in newline.

```
PRINTLN “Hello world”      PRINTLN var1 + var2
PRINTLN var                PRINTLN var + 12
PRINTLN 1234
```

b) ***Supported control structures***

1. Sequence: This structure represents a program with no alternative options, such as selection and iteration control structure. It means the structure processes the pseudocode one after the other following the top-to-bottom approach.

This construct is represented as a sequence of pseudocode statements:

```
Syntax:                 Examples:
Statement1 ;        width = PROMPT(“Enter  the width ”)
Statement2 ;        length = PROMPT(“Enter  the lengh ”)
Statement3 ;        Calculate area=width * lenght
Statement4 ;        PRINT “The area is”
Statement5;         PRINT area
```

ISSN 1533-9211

DOI 10.5281/zenodo.8241305

2. Selection: The selection control structure allows the user to create pseudocode with two or more alternative options. This option depends on whether the condition is true or false. If the condition is true, the system will execute the TRUE statement. Otherwise, the system will execute the FALSE statement. Relational operators are used in the condition to test the variable's value as to whether it is true or false. Under this category are Simple IF with null ELSE, Simple IF-ELSE, Ladderized IF, Nested If and Combined IF using logical AND and OR.

2.1 Simple IF with null ELSE: This selection control structure returns only a statement or results if the condition evaluates to true. If the condition is false, the program will not return any result, hence the name null else.

Simple IF null ELSE Syntax:

```
IF condition
  Statement or statements;
END
```

2.2. Simple IF-ELSE: The structure uses the usual IF ELSE format. The construct works by evaluating the condition and finding whether the result of the evaluation is true or false. If the result is true, the statement under the IF statement is executed. Otherwise, the program will execute the statement under the ELSE statement.

Simple IF-ELSE Syntax:

```
IF condition
    Execute the statement if the condition result is true
END
ELSE
   Execute the statement if the condition result is false
END
```

2.3. Ladderized IF: This structure has multiple IF statements. This construct would test several conditions to find the appropriate option starting from the first condition. If the construct finds the right option, it will execute the statement under it and skips all subsequent conditions. Otherwise, the evaluation process reaches the Else statement and executes the statement under it. The Else statement serves as a default action if none of the conditions before it are met.

Ladderized IF Syntax:

```
IF condition
        Execute the statement if the condition result is true
END
```

The Seybold REPORT
ISSN 1533-9211
DOI 10.5281/zenodo.8241305

```
ELSE IF condition
    Execute the statement if the condition result is true
END
ELSE IF condition
    Execute the statement if the condition result is true
END
ELSE
    Execute the statement if the condition result is false
END
```

2.4. Nested IF: This structure is when the user must have several conditions that must be true before a statement or group of statements is executed. In nested If, there are outer IFs and inner IFs.

Nested IF Syntax:

```
IF Condition
IF condition
    Execute the statement if the outer and inner condition result is true
END
ELSE
    Execute the statement if the condition result if the inner condition is false
END
END
ELSE
    Execute the statement if the condition result if the outer condition is false
END
```

2.5. Combined IF using logical AND and OR: This structure has several conditions connected with logical operators AND or OR. The condition which includes the AND (&&) statement has the following results:

Condition 1	Condition 2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

The Seybold REPORT
ISSN 1533-9211
DOI 10.5281/zenodo.8241305

On the other hand, the OR (||) statement has the following results

Condition 1	Condition 2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Combined IF Syntax:

```
IF (condition1 && condition2)
        Execute the statement if the condition1and condition2 is true
END
ELSE
        Execute the statement if the condition1and condition2 is false
END
```

B. Repetition:

This structure includes loops or iterations. This construct executes a statement or group of statements repeatedly as long as a specific condition evaluates to true. If the condition becomes false, the loop will terminate, and the program will end its execution. The while statement tests the condition first, whether true or false, before executing a statement or group of statements.

WHILE Syntax:

```
Initialization;
WHILE  (condition)
            Statement(s);
            Update;
END
```

C. Array:

It is a series of homogenous elements that has a similar data type and is stored continuously under a single name or variable name (Chaudhary, 2014). Programmers, while using an array, must understand and follow the rules for using it in the program. The array includes the following data.

- Element – Any value inside an array is referred to as element.
- Index − Each element stored in an array has an address which points where the element is stored. It starts with a 0 and ends with n-1.

Array Syntax:

ISSN 1533-9211

DOI 10.5281/zenodo.8241305

Array Declaration:

array_name = [element1 , element2 , element3 , element4];

Array Access:

array_name[index] = element;

D. Function:

It is a reusable block of code that performs a specific task or set of related tasks. Programmers can call these reusable codes anywhere in the program or whenever they need to execute them. To use a function, one must define the function's name, optional return type, and parameters. The program can then use the name to invoke the function whenever needed.

Function Syntax:

Function Declaration:

```
        FUNC function_name (parameter1, parameter2, parameterN)
        Statement(s)
END
```

E. Supported Operators:

These mathematical, relational or logical operators can be used in the program.

Arithmetic Operator

Description	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/

Relational Operators

Description	Symbol
Greater than	>
Less than	<
Greater than or equal to	>=
Less than or equal to	<=
Equal to	= =
Not equal	! =

Logical Operator

The Seybold REPORT

ISSN 1533-9211

DOI 10.5281/zenodo.8241305

AND	&&
OR	||\

As shown in Data Flow Diagram Level 1, the Pseudocode Interpreter shows the program's three processes, one external entity and one data store. Based on the diagram, lexical analysis is the first phase of a compiler (refer to process 1). The system used the lexical analyzer to separate the pseudocode lexemes from one another. The system achieved this by ignoring whitespaces and single-line and multiple-line comments. If the lexical analyzer finds a token invalid, it generates and sends an error message (lexically invalid) back to the user.

The lexical analyzer works closely with the syntax analyzer (process 2). It reads the character streams from the pseudocode source, checks for the legal token by retrieving and matching the token type in the data store and passes the data to process 2 (the syntax analyzer). Additionally, a lexical analysis works as a text scanner. The lexical analyzer reads the entered pseudocode, converts each group of characters into meaningful lexemes, stores it on a data structure, creates the token, places it in an abstract tree, and checks whether each line follows the specified pseudocode format. The lexer continues to read up to the last character. Each character is combined and categorized as an OPERATOR, KEYWORD, NUMBER, STRING, or similar.

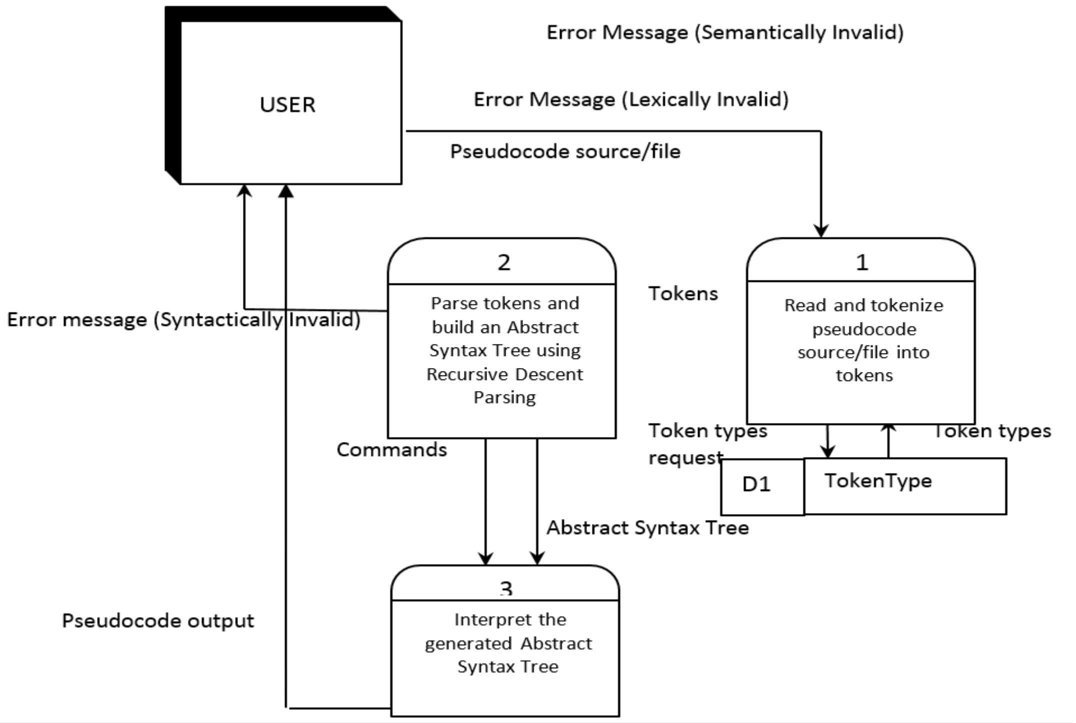
Error Message (Semantically Invalid)
USER
Error Message (Lexically Invalid)
Pseudocode source/file
2
Parse tokens and build an Abstract Syntax Tree using Recursive Descent Parsing
Tokens
1
Read and tokenize pseudocode source/file into tokens
Error message (Syntactically Invalid)
Token types request
Token types
Commands
D1
TokenType
Abstract Syntax Tree
3
Interpret the generated Abstract Syntax Tree
Pseudocode output

Figure 1: Data Flow Diagram Level 1 of the Pseudocode Interpreter

ISSN 1533-9211

DOI 10.5281/zenodo.8241305

Evaluation and Respondents of the Study

The respondents evaluated the system using black box testing. This testing is done with the tester only having access to the inputs and expected outputs of the system. The goal is to verify whether the system behaves correctly based on different inputs and to identify any discrepancies between the expected and actual outputs. It primarily concerns validating the system's functionality, usability, and adherence to specified requirements or business logic (Sholeh et al., 2021).

The researchers asked the evaluators to rate the system based on the software standards criteria of ISO 9126. Abra et al. (2008) published an instrument for evaluating software products based on ISO 9126. The respondents evaluate the software based on its unique set of criteria for each ISO 9126 software characteristic, namely Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability.

The researchers used purposive sampling in selecting the respondents of the study. The researchers identify specific criteria or characteristics important for their research objectives and use them to select participants who meet those requirements. The researchers have chosen 30 respondents, including ICT professionals, faculty members handling the course, and students of the course as study respondents. Respondents had to answer the survey form, and the researchers considered their suggestions and recommendations for enhancing the software.

Data Processing and Statistical Treatment

The researchers used Microsoft Excel 2007 to tabulate the data collected from the survey. The researchers used the weighted Mean to determine the conformance of the system features to the requirements of the users or the respondents and Standard Deviation (SD) to assess the variability of each answer.

The researchers use the following rating scale to interpret the evaluation result. The rating scale is as follows: 4.21 to 5.00 Excellent, 3.41 to 4.20- Very Satisfactory, 2.61 to 3.4- Satisfactory, 1.81 t0 2.60-Good and 1.00 to 1.80 -

RESULTS AND DISCUSSION

The ICT respondents, which include programmers, developers, IT personnel, educators, and students, were asked to evaluate the performance of the Pseudocode Interpreter using the six quality standards established by ISO 9126. The results of the respondents' evaluation were tabulated and analyzed.

The evaluation result for Pseudocode Interpreter, as shown in Table I, was rated with an overall average mean of 4.36, is described as "Excellent", and the standard deviation (SD) result is 0.58.

The Seybold REPORT
ISSN 1533-9211
DOI 10.5281/zenodo.8241305

Table I: ISO 9126 Software Evaluation Standard of the Software

Characteristics	Average Mean	Description	Standard Deviation
Functionality	4.27	Excellent	0.65
Reliability	4.36	Excellent	0.62
Usability	4.56	Excellent	0.54
Efficiency	4.46	Excellent	0.51
Maintainability	4.33	Excellent	0.55
Portability	4.17	Very Satisfactory	0.60
Overall Average Mean	4.36	Excellent	0.58

The Functionality of the system has a mean value of 4.27, described as "Excellent". According to Chirinos et al. (2003), the rating implies that the system meets the functional requirements outlined during its design and development. It performs the tasks and functions as designed, and the features and capabilities specified are present and working as expected. These results show that the software met the requirement for the functionality characteristics of the ISO 9126 standard. The result shows that the system could accept pseudocode, read and parse each statement and check whether the pseudocode follows the pre-defined format, and the desk checking tool displays the result.

The system's reliability has a mean of 4.36 and an SD of 0.6, corresponding to "Excellent". The result means that the software met the requirement for the reliability characteristics of the ISO 9126 standard. The result means the software consistently performs its intended functions correctly and accurately over time without unexpected failures or errors. It likewise implies that the study ensures that the software can be trusted to operate as expected and deliver consistent results. Also, the respondents agreed that the consistency and accuracy in the storage and retrieval of the related libraries work smoothly. The stored records of the previous pseudocode source are readily accessible. The respondents also evaluated the consistency and accuracy of the software's user interface. The results imply that all the user interface elements of the software function properly.

The Usability of the system was rated with a mean of 4.56 and SD of 0.54 and described as "Excellent". According to Bosch et al. (2003), the software is easy to understand, learn, and use because its interface is well-designed and easy to follow. The result also implied that the software is easy to use since instructions for using the application are visible and readily available. In terms of understandability, the user can easily read the application, and the user prompts and messages are available to help students manipulate the software.

The system's efficiency was rated with a mean of 4.46 and SD of 0.51 and described as "Excellent". This result implies that regarding Resource Behavior, the "Pseudocode Interpreter" was perceived by the respondents to have high performance in terms of minimizing resource consumption and providing optimal responsiveness in response to user requests. It also shows that the system avoids unnecessary or excessive resource consumption and strives to balance performance and resource usage.

The system's maintainability was rated with a mean of 4.33 and SD of 0.55 and described as "Excellent". The result signifies that the researchers designed, developed, and organized to

ISSN 1533-9211

DOI 10.5281/zenodo.8241305

facilitate easy, cost-effective maintenance and support as perceived by the respondents. The result also shows that the modular structure of the software allows for easier understanding, debugging, and modification of specific parts of the software without impacting the entire system. Moreover, the result shows that the system helps users quickly identify, report, and handle errors.

The system's Portability was rated with a mean of 4.17 and SD of 0.60 and described as "Very Satisfactory". The result implies that the software can be easily deployed across different platforms or environments without significant modifications or adaptations. It also implies that the researchers designed the software to run efficiently and consistently on various operating systems, hardware architectures, or software environments, providing users flexibility and ease of use. Since the researchers designed the system as online and mobile-based, it can be conveniently accessed anytime using a computer or mobile device.

The overall result means that the software is functional and it conforms to the standards based on ISO 9126. It also shows that the respondents' responses are consistent, or the majority of the ratings are the same.

CONCLUSION AND RECOMMENDATION

The researchers supported the evaluation of the system based on ISO 9126 and can conclude that the software "Pseudocode Interpreter" is ready for implementation. The result shows that the software can accept pseudocode based on the standards written by Robertson (2006). The software could accept structured theorems such as sequence, selection, iteration, and recursive functions. The software could also perform lexical analysis, create a valid token from the lexemes and generate results based on the predefined rules. The system can also accept and parse mathematical operators such as arithmetic, logical, and relational. The system can also accept inputs for arrays and functions and perform calculations based on these inputs.

The application can analyze and display the desk-check table of the pseudocode, and the respondents rated the system as excellent in terms of its functionality, reliability, usability, efficiency, maintainability and Very Satisfactory in terms of portability. Thus, the researchers can conclude that the software conforms to the ISO 9126 software quality standards based on the respondents' evaluation.

The researchers recommend that the study will be subjected to further research and development by the ICT students who will conduct a related study. Moreover, the researchers recommend that the software may be enhanced by adding new features, such as the capability to draw and check the flow chart of the given pseudocode, since this is also a part of the lesson for Programming Logic Formulation.

ISSN 1533-9211

DOI 10.5281/zenodo.8241305

References

1. Abran, A., Qutaish, R.A., Desharnais, J., & Habra, N. (2008). Software quality measurement: Concepts And approaches. Institute of Chartered Financial Analysts of India, 2008.

2. Bosch, J., & Lundberg, L. (2003). Software architecture – engineering quality attributes. Journal of Systems and Software, 66(3), 183–186.

3. Chaudhary, H. (2014). C Programming: The ultimate way to learn the fundamentals of the C language. First MIT- Createspace Inc. O-D-Publishing, LLC.

4. Dennis, A., Wixom, B., & Roth, R. (2012). System Analysis and Design (5th ed.). John Wiley & Sons.

5. Department of Education. (n.d.). Technology and livelihood education (TLE) and Technical-Vocational-Livelihood (TVL) track. K to 12 Basic Education Curriculum. https://www.deped.gov.ph/k-to-12/about/k-to-12-basic-education-curriculum/technology-and-livelihood-education-tle-and-technical-vocational-livelihood-tvl-track/

6. Farell, J. (2015). Programming Logic and Design Eight Edition. Cengage Learning

7. harvard.edu (2006). Extended Backus Naur Form (EBNF). https://www.arp.harvard.edu/eng/das/manuals/EBNF.html

8. Hsiao, T., Chiang, Y., Chen, T., Chang, C. & Chen, C. (2022). Students' performances in computer programming of higher education for sustainable development: The effects of a peer-evaluation system. Frontiers in Psychology https://www.frontiersin.org/articles/10.3389/fpsyg.2022.911417/full

9. Jones, G. B. (2019, December). The struggles experienced by first year computer programming students at the University of South Africa. https://www.dpublication.com/wp-content/uploads/2019/12/4-440.pdf

10. Losavio, F.. Chirinos. L, Lévy, N. & Ramdane-Cherif, A. (2003). Quality characteristics for software architecture, The Journal of Object Technology, 2(2), 133.

11. Peisert, S. (2002). Lexers. https://web.cs.ucdavis.edu/~peisert/teaching/cse131a/lecture_slides/131a_lecture1b.pdf

12. Ramat, M., Shahran , S., Latih, R., Yatim, N., Zainal, N., & Rahman, R. (2011). Major problems in basic programming that influence student performance. Procedia - Social and Behavioral Sciences, 59, 287-296.

13. Rillo, R., & Alietio, E. (2018). Indirectness markers in Korean and Persian English essays: Implications for teaching writing to EFL learners. Journal of English as International Language, 13(1), 1-14. https://files.eric.ed.gov/fulltext/ED596726.pdf

14. Robertson, L. (2006). Simple Program Design: A Step-by-Step Approach (5th ed.). Thomson Nelson Australia Pty Limited.

15. Muhammad S., Gisfas, I., Fauzi, C., & Fauzi, M. (2021). Black Box Testing on ukmbantul.com Page with Boundary Value Analysis and Equivalence Partitioning Methods. Journal of Physics: Conference Series, 1823(1), 012029. https://iopscience.iop.org/article/10.1088/1742-6596/1823/1/012029

16. Official Gazette of the Republic of the Philippines. (n.d.). The K to 12 basic education program. https://www.officialgazette.gov.ph/k-12/