Source Code Metrics for Programmable Logic Controller (PLC) Ladder Diagram (LD) Visual Programming Language

Lov Kumar
lov.kumar@in.abb.com

Raoul Jetley
raoul.jetley@in.abb.com

Ashish Sureka
ashish.sureka@in.abb.com

ABB Corporate Research Center
Bangalore, India

ABSTRACT

The IEC 611131-3, an open international standard for Programmable Logic Controllers (PLC) defines several domain specific languages for industrial and process automation. Domain specific languages have several features, programming constructs and notations which are different than general purpose languages and hence the traditional source code metrics for general purpose programming languages cannot be directly applied to domain specific languages for writing control programs in the area of industrial automation. We propose source code level metrics to measure size, vocabulary, cognitive complexity and testing complexity of a visual Programmable Logic Controller (PLC) programming language. We present metrics for Ladder Diagram (LD) programming language which is one of the five languages defines in the IEC 61131-3 standard. The proposed metric is based on factors like number of distinct operators and operands, total number of operators and operands, number of rungs, weights of different basic control structures, structure of the program and control flow. We apply Weykur's property to validate the metrics and evaluate the number of properties satisfied by the proposed metric.

Keywords

Source Code Metrics, Programmable Logic Controller (PLC) Programming Languages, Software Complexity Metrics, Software Size Metrics, Structured Text, Ladder Diagram (LD), Halstead Metrics, Cyclomatic Complexity

1. RESEARCH MOTIVATION AND AIM

Programmable Logic Controller (PLC) are control systems used for factory and industrial automation of electromechanical processes. PLCs are programmed using domain specific languages [5][10]. The International Electrotechnical Committee (IEC) developed a standard called as IEC 61131-3 which defines the basic programming elements, syntactic and semantic rules for text-based and graphical or visual programming languages for programming PLCs. IEC 61131-3 defines graphical languages like Ladder diagram (LD) and Function block diagram (FBD) and text-based languages like Structured text (ST) and Instruction list (IL). A Domain-Specific Language (DSL) like a Ladder Diagram (LD) is specialized to a particular application domain and hence has specialized features as well as programming constructs not present in general purpose programming languages [5][10]. Similarly, a DSL does not contain all the features of a general purpose language as it is designed to solve problems in a specific domain unlike general purpose languages which are designed to solve problems across several domains.

Software size and complexity metrics are used for cost and effort estimation, manpower allocation, maintainability and testability projections, program and developer evaluation [2][6]. Several size (Lines of Code, Halstead Metrics) and complexity metrics (McCabe Cyclomatic Number) have been proposed in the literature [2][6]. However, popular metrics like Halstead Metric and McCable Cyclomatic Complexity cannot be directly applied to domain specific text-based and visual PLC programming languages due to different and specialized syntax, programming paradigm, basic elements and programming constructs of DSLs. Size and Complexity metrics for PLC programming languages used in the domain of factory and industrial automation is an area which is relatively unexplored. The work presented in this paper is motivated by the need to define practical and rigorous metrics for PLC domain specific languages. The specific aim of our study is to define size, vocabulary, cognitive complexity and testing complexity of Ladder Diagram (LD) which is a graphical programming language defined in the IEC 61131 Standard.

2. RELATED WORK AND RESEARCH CONTRIBUTIONS

Gharieb et al. propose metrics to measure the software quality for Programmable Logic Controllers (PLCs) especially in ladder programming [3]. Their proposed quality metrics involves the criteria of simplicity, re-configurability, reliability, and flexibility [3]. Nair et al. present a methodology to define metrics for domain specific languages [9]. Using their proposed methodology, they define a set of product metrics that can be used for managing the software project development using IEC 61131-3 languages [9]. Lucas et al present methods of measuring the size and complexity of PLC programs in different logic control design methodolo-

WETSOM '16 Austin, Texas USA

DOI: 10.475/123_4

gies [7]. Cardoso et al. present a survey of process complexity metrics in the domain of business process modelling [1]. They describe several approaches to adapt known software metrics proposed by researches to business processes analysis [1]. Gruhn et al. discusses how existing research results on the complexity of software can be extended in order to analyze the complexity of business process models [4]. Waters et al. describes a method for creating tools to calculate software metrics for ladder logic, specifically Rock-well Automation's implementation of ladder logic for its ControlLogix family of PLCs, Import-Export language version 2.6 [11].

There are several differences between previous studies and the work presented in this paper. The size and complexity metric proposed by Nair et al. is based on parameters like time required to configure a standard Program Organization Unit (POU) by an average resource, number of additional operands to expand POUs, number of physical I/O and effort spent on implementing modules [9]. We believe that metrics based on time and effort required to develop and understand a software can be subjective, contextual, based on psychological factors and hard to generalize. Lucas et al. present a metric to measure the size of a module as a function of number of operations in a module [7]. The metric proposed by Lucas et al. does not take into account the cognitive weights of basic control structures and elements as well as the control flow of a program. The similarity between the work by Cardoso et al. and Gruhn et al. is that the both the studies are aimed at defining size and complexity metrics for visual or graphical programming languages [1] [4] but the different is that our work is focused on Ladder Digaram (LD) whereas the work by Cardoso et al. and Gruhn et al. is on business process modeling (different syntax and constructs).

Novel Research Contributions: The study presented in this paper is the first work on defining vocabulry, size, cognitive complexity and testing complexity of Ladder Diagram (LD) programs using factors like number of distinct operators and operands, total number of operators and operands, number of rungs, weights of different basic control structures, structure of the program and control flow. Furthermore, we present an application of Weykur's property to validate the metrics and evaluate the number of properties satisfied by the proposed metric.

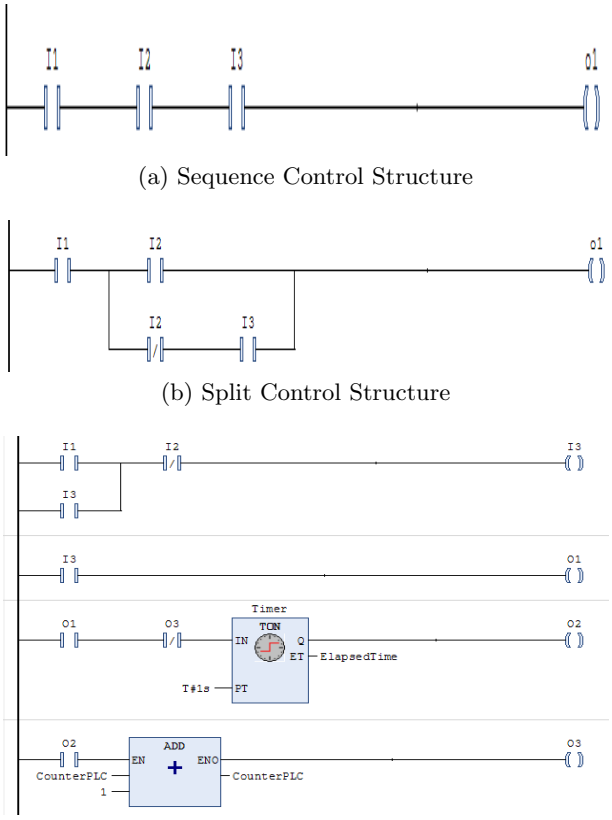
3. PROPOSED METRICS

3.1 Vocabulary (VC)

We define program vocabulary as the total number of distinct operators and operands used in a given Ladder Diagram. An operator in a programming language is defined as a special symbol which performs specific operations on one or multiple operands and returns result. We consider operand as a symbol used to represent a ladder diagram element. For example, an input contact, output coil, counter, timer and a predefined add function are operators. In Figure 1a, the VC value is 6 because there are 2 distinct operators and 4 distinct operands ($I1$, $I2$, $I3$ and $O1$). Similarly the VC value for the LD in Figure 1b and 1c is 7 and 15 respectively.

3.2 Size (SZ)

We define size as the number of executable basic control



(a) Sequence Control Structure

(b) Split Control Structure

(c) Timer and Pre-Defined Library Function Control Structure

Figure 1: Examples of Different Control Structures

structures in a Ladder Diagram program. The proposed metric for size is similar to the LOC (Lines of Code) metric used in general purpose text-based programming languages which counts the number of executable statements in a program. For example, the input contact and output coil in a LD is regarded as equivalent to executing a statement in a general purpose programming language. The Vocabulary and Size metrics does not take into account the cognitive effort required to understand the basic control structures and the control flow of the program. Figure 1 displays examples of three Ladder Diagrams of different size and complexity. The size of the LD in Figure 1a is 4 because there are 3 inputs and 1 output element. The size of the LD in Figure 1b is 5 because there are 4 inputs and 1 output elements (hence there are 5 executable elements). Similarly, the size of the LD in Figure 1c is 13.

3.3 Cognitive Complexity (CC)

We define Cognitive Complexity (CC) as a metric to measure how easy or difficult it is to understand and comprehend a given Ladder Diagram program. CC is a function of the size of the program, types and numbers of basic control structures used and the structure of the program. We propose Cognitive complexity (CC) of a ladder diagram as Equation 1.

$$CC = (N_i + N_o) * W_c \quad (1)$$

Where N_i and N_o represents the number of distinct input variables and output variables respectively. W_c represents

Table 1: Basic Control Structures and their Cognitive Weight

Category	**BCS**	**Weight**
Sequence	SEQ	1
Branch	Split	2
	JUMP	2
Iteration	For	3
	Timer	3
	counter	3
Embedded Component	Arithmetic Function	2
	Control function	2
	Data comparison	2
	Data movement	2
	Data conversion function	2
	Binary function	2
	Sequencer function	2
Concurrency	Time interrupt	4
	Fault interrupt	4
	Input driven Interrupt	4

the total cognitive weight of the given ladder diagram. The total cognitive weights (W_c) of a ladder diagram is calculated using Equation 2.

$$W_c = \sum_{i=1}^{N_r} W_i \tag{2}$$

Where W_i is the weight of rung i and N_r is the number of rungs in the given ladder diagram. Table 1 shows the basic control structures and their cognitive weight of PLC ladder diagram. We assign the cognitive weight for the various control structures in Table 1. The weight for a Sequence control structure is lower than operations like branch, split, iterations and concurrency because a sequence structure is easier to understand in compasrion to other structures. Each rungs of the ladder diagram contains M number of basic control structure. Hence, the modified form of Equation 2 can be written as Equation 3:

$$W_c = \sum_{i=1}^{N_r} \sum_{j=1}^{M} W_{i,j} \tag{3}$$

For a rung in a ladder diagram, a basic control structure (such as a sequence or split) may consists of P layers of nested basic control structures which increases the cognitive complexity of the program. Hence, to incorporate nested control structure, we extend Equation 3 to Equation 4:

$$W_c = \sum_{i=1}^{N_r} [\prod_{k=1}^{p} \sum_{j=1}^{M} W_{i,j}] \tag{4}$$

Figure 1 shows examples of three ladder diagrams with different control structures. Figure 1a shows the example of a sequence control flow structure as all the input contacts and output coils are ordered in a single sequence. Figure 1b shows the example of a split control flow structure since the input contact $I2$ is placed in parallel to $I2$ and $I3$. Figure 1c shows the example of a timer, a predefined function (add) and some inputs and outputs.

Figure 2 shows two Ladder Diagram programs. The number and types of basic control structures in both the diagrams are same. There are only two distinct operators and 7 distinct operands used in the ladder diagrams illustrated in Figure 2. Furthermore, both programs have same number of independent paths. However, the structure of both the programs are different. The Ladder Diagram in Figure 2b is more cognitively complex as compared to the Ladder Diagram of Figure 2a due to the nesting structure and depth. Figure 2a shows an almost linear control flow of split operation whereas in Figure 2b, there are several nesting split operations. The CC value for both illustrative examples are computed as follow:

Figure 2a: The CC value of the ladder diagram in Figure 2a is calculated using Equation 1. The N_i and N_o values are 7 and 1 respectively. Hence, CC is $(7+1) * W_c$. The ladder diagram does not contain any nesting control flow structure. Thus, the total cognitive weights (W_c) is calculated as follows using Equation 3.

$$W_c = \sum_{i=1}^{N_r} \sum_{j=1}^{M} W_{i,j} \tag{5}$$

The ladder diagram contains only one rung (N_r=1) and 4 control flow structures (one sequence and 3 split). The W_c value of the ladder diagram is: $W_c = 1+2+2+2 = 7$. After computing cognitive weights (W_c), the cognitive complexity (CC) of the ladder diagram is: $CC = (7+1)*W_c = 8*7 = 56$.

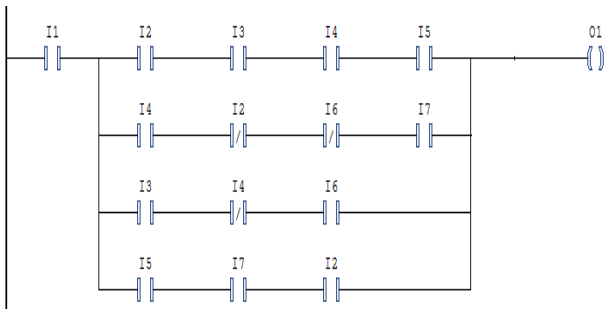
Figure 2b: We calculate the CC value of the ladder diagram using Equation 1. The N_i and N_o values of the program are 7 and 1 respectively which is the same as the ladder diagram shown in Figure 2a. After applying the formula in Equation 1, we get the CC value of the program as $(7+1)*W_c$. The ladder diagram in Figure 2b contains nested control flow structure. Thus, the total cognitive weights (W_c) calculated using Equation 4 is as follows:.

$$W_c = \sum_{i=1}^{N_r} [\prod_{k=1}^{p} \sum_{j=1}^{M} W_{i,j}] \tag{6}$$

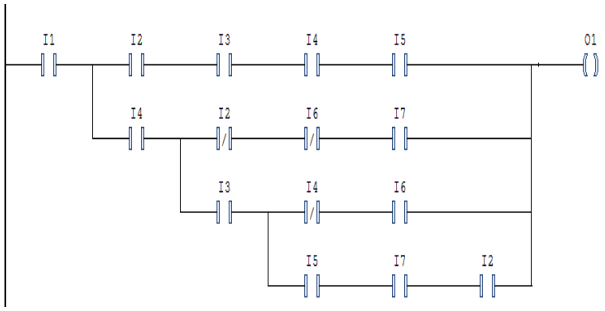
The program contains one run and four basic control flow structures (1 sequence, 1 split and 2 nested splits). Hence the W_c value of ladder diagram is: $W_c = 1 + 2 * (1 + 2 * (1 + 2)) = 15$. After computing the cognitive weights (W_c), the cognitive complexity (CC) of the program is: $CC = (7 + 1) * W_c = 8 * 15 = 120$.

Size Vs. Cognitive Complexity: Using the worked-out example for the ladder diagrams displayed in Figure 2, we conclude that two programs of similar size can have different cognitive complexity. The size of the both the programs is 16 whereas the Cognitive Complexity of one program is 56 and the other is 120. The cognitive complexity of the second program is more than double of the first program despite similar sizes.

Vocabulary Vs. Cognitive Complexity: The vocabulary for both the ladder diagram programs of Figure 2 is 10 whereas the cognitive complexity values are significantly



(a) Ladder Diagram with Simple Control Structure



(b) Ladder Diagram with Nested Control Structure

Figure 2: Ladder Diagram with Simple and Nested Control Structures

different.

3.4 Testing Complexity (TC)

We define Testing Complexity (CC) as a measure of the sum of all possible control flows for each rung in the LD. The TC is measure of testability (different than the understandability of the design) which is correlated to CC but still different and incorporates factors not captured in CC. TC of a ladder diagram is computed using Equation 7.

$$TC = \sum_{i=1}^{N_r} TC_i \tag{7}$$

Where TC_i is the testing complexity of rung i and N_r is the number of rungs in the ladder diagram. A rung which contains more than one control flow path can be classified as a *split* component. In case of single split component, atleast one and at-most two control paths are possible. Hence, for every split component, $2^2 - 1$ is added to TC. Each rungs may contain M number of sequential split component. Hence the Equation 7 can be extended to Equation 8:

$$TC = \sum_{i=1}^{N_r} \sum_{j=1}^{M} TC_{i,j} \tag{8}$$

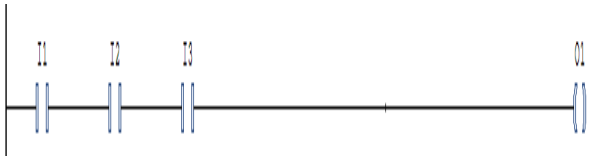
Where $TC_{i,j}$ is the testing complexity of j^{th} *split* of i^{th} rung. One basic split may consists of P number of nesting split component. Hence we extend the Equation 8 as Equation 9

$$TC = \sum_{i=1}^{N_r} \sum_{j=1}^{M} 2^{P_j+1} - 1 \tag{9}$$

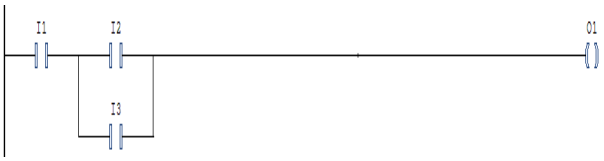
Figure 3 shows two PLC ladder diagram P and Q. In P, there is only one control flow path (CFP) between input to output. In Q, there is one split component ($I2$ and $I3$ in parallel). Hence the total number of control flow path in Q is $2^2 - 1 = 3$. The control flow for P is: I1=True, I2=True, I3=True → O1=True. There are three control flows for Q:

1. I1=True, I2=True, I3=False → O1=True
2. I1=True, I2=False, I3=True → O1=True
3. I1=True, I2=True, I3=True → O1=True

Size Vs. Testing Complexity: Using the worked-out example for the ladder diagrams displayed in Figure 3, we conclude that two programs of similar size can have different testing complexity. The size of the both the programs is 4 whereas the Testing Complexity Complexity of one program is 1 and the other is 3.



(a) P



(b) Q

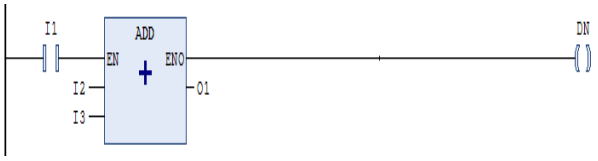
Figure 3: Examples of PLC Ladder Diagram: (a) Sequential (b) Split

Vocabulary Vs. Testing Complexity: The vocabulary for both the ladder diagram programs of Figure 3 is 6 whereas the testing complexity one program is 1 and other is 3. From this analysis, we conclude that two programs of similar vocabulary can have different different testing complexity.

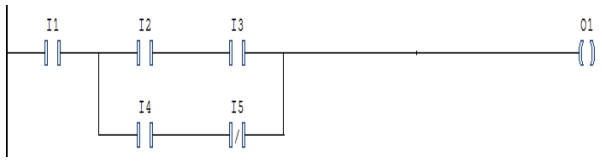
Cognitive Complexity Vs. Testing Complexity: Figure 4 shows two PLC ladder diagram P and Q. The cognitive complexity for both the ladder diagram is 15 whereas the testing complexity of one program is 1 and other is 3. Using this example, we conclude that two ladder diagram of similar cognitive complexity i.e., difficult to understand can have different testing complexity.

4. WEYUKER'S PROPERTIES ANALYSIS

Weyuker define a formal list of properties for software metrics which has been used by several researchers to evaluate their proposed software metrics [12]. In this Section, we present our analysis of Weyuker's properties on our metrics and present our examination of the number of Weyuker properties satisfied. The objective of analyzing Weyuker's properties is to validate our proposed metrics is to user a



(a) P



(b) Q

Figure 4: Examples of PLC Ladder Diagram (a) Pre-defined Function (b) Split

well-known method which has played an important role in evaluating software complexity measures.

Property 1: $(\exists P)(\exists Q)(CC(P) \neq CC(Q))$

The property is an essential requirement and states that there are programs P and Q such that the complexity of P is not equal to the complexity of Q. The property requires that the metric (CC in our case) should not produce the same value for every program. Figure 3 shows two PLC ladder diagrams: P and Q. The CC value of P and Q are $1 * (3 + 1) = 4$ and $(1 + 2) * (3 + 1) = 12$ respectively. The example demonstrates that there exists two programs which have different CC value. Hence, the CC metric satisfies Property 1.

Property 2: Let c be a non-negative number. Then there are finitely many programs of complexity c.

In PLC ladder diagram, the number of distinct elements are fixed, and all program uses this fixed number of component to draw their ladder diagram. At one instance, the component will get repeated and resulting ladder diagrams have same number of inputs and output element, but the possible number of structure of the ladder diagrams are infinite, that give infinite value of W_c. Hence, We conclude that Property 2 is not satisfied for CC metric.

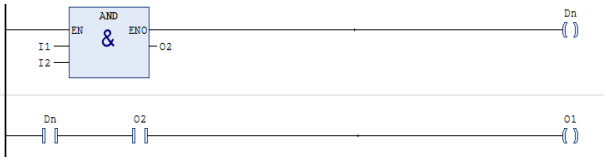
Property 3: There are distinct programs P and Q such that $(CC(P) = CC(Q)$

The property ensures that not every ladder diagram program maps to a distinct CC value. Property 3 requires that there are at least two different programs that yield the same measurement value. Figure 4 shows two distinct PLC ladder diagrams: P and Q. The CC value of P and Q are $(1 + 2) * (3 + 2) = 15$ and $(1 + 2) * (4 + 1) = 15$ respectively. The example demonstrates that there exists two distinct program having the same CC value. Hence CC satisfy Property 3.

Property 4: $(\exists P)(\exists Q)$ (P=Q & $CC(P) \neq CC(Q)$)

This property ensures that at-least two different ladder diagram programs with equal functionality yield distinct CC measurement values. Two programs can be expressed differently even though they have the same functionality but their complexity measure can vary depending on the expression. Figure 5 shows two equivalent PLC ladder diagrams (in-terms of the functionality): P and Q. In both programs, the output $O1$ should be $true$ when input $I1$ is $True$ and input $I2$ is $true$, otherwise output should be $false$. The CC value of P and Q are $(1)*(2+1) = 3$ and $(2+1)*(2+3) = 15$ respectively. The example demonstrates that, two equivalent program in-terms of functionality but different expressions have different CC value. Hence, CC satisfies Property 4.

(a) P



(b) Q

Figure 5: Examples of PLC Ladder Diagram showing Sequential Structure and Pre-Defined Function Element

Property 5: $(\forall P)(\forall Q)$ $(CC(P) \leq CC(P;Q)$ & $CC(Q) \leq CC(P;Q))$

This property pertains to the monotonicity characteristic and ensures that the CC measurement value only increases or grows and never decreases when composing a program body by adding another block before the original block or after the original block. We demonstrate the satisfy-ability of Property 5 using concatenation. If we concatenate two PLC ladder diagrams, the number of rungs is always greater or equal to number of rungs of largest ladder diagram.

Property 6a: $(\exists P)(\exists Q)(\exists R)$ $(CC(P) = CC(Q)$ & $CC(P;R) \neq CC(Q;R))$

Property 6b: $(\exists P)(\exists Q)(\exists R)$ $(CC(P) = CC(Q)$ & $CC(R;P) \neq CC(R;Q))$

This property pertains to the interaction characteristics between program bodies. Let us say we have two program bodies (B and C) with equal CC measurement values. Let us say we have another program body (A) which interacts with the two program bodies (B and C) having equal measurement values. The program body A may interact differently with B and C. This property requires the measure to be responsive to this kind of interaction (consider the interaction also). Figure 6 shows five PLC ladder diagrams: P, Q, R, concatenation of P and Q (P;R), and concatenation of Q and R (Q;R). The CC value both ladder diagram P and Q are same i.e., $1*(2+1) = 3$. The CC values after concatena-

tion of R with P and Q are $CC(P;R) = (1+2)*(3+1) = 12$, and $CC(Q;R) = (1+1)(3+2) = 10$. The example demonstrates that CC satisfy Property 6.

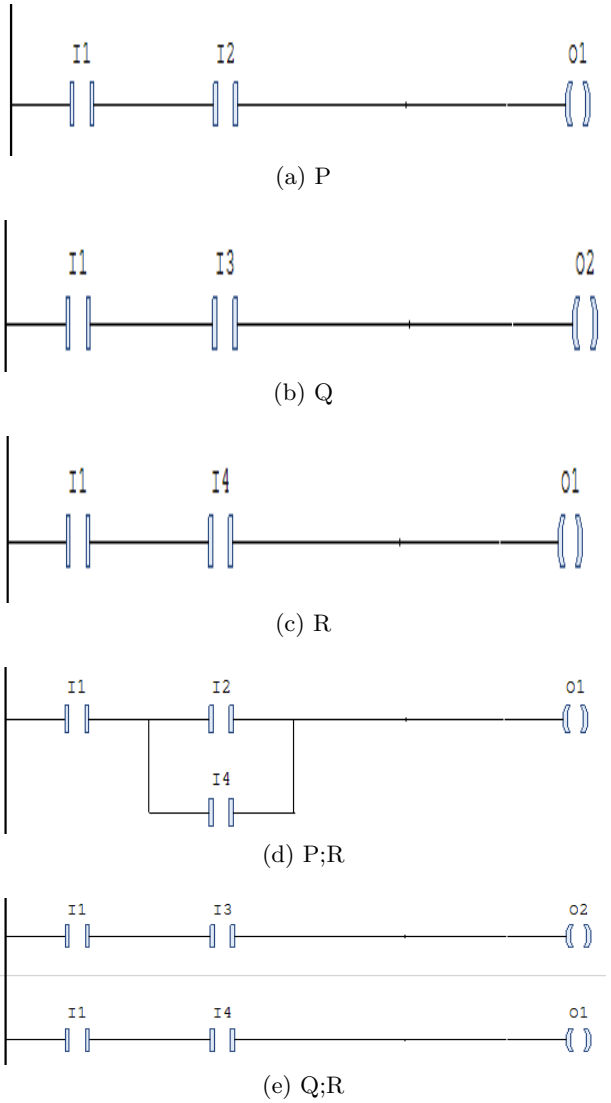


Figure 6: Examples of PLC Ladder Diagram showing simple structure and concatenation

Property 7: There are program bodies P and Q such that Q is formed by permuting the order of the statements of P, and $(CC(P) \neq CC(Q))$

Having the same intent as Property 6, this property also pertains to interaction between program bodies. The property ensures that potential interaction is evaluated by the measure and is sensitive to the interaction. CC satisfies this property because the permutation of control flows can result in different control flows structure (resulting in change in structure), hence making the CC value different.

Property 8: If P is a renaming of Q, then $(CC(P) = CC(Q))$

CC value of Ladder diagram depends on the control flow structure of the program as well as the cognitive weights of the various elements used in the program. The renaming of various elements of the ladder diagram does not change the structure and number and type of distinct operators. Hence, CC satisfies Property 8 (because variable names are not be part of a structural analysis while computing the cognitive complexity of a ladder diagram program).

Property 9: $(\exists P)(\exists Q)(CC(P) + CC(Q) < CC(P;Q))$

This property ensures that the CC metric should result in a possible increase in complexity if an interaction of two concatenated bodies occurs. The property requires the existence of two program bodies whose concatenation has a cognitive complexity measurement value which is more than the sum of its constituent parts. Figure 7 show the three PLC ladder diagrams: P, Q, and concatenation of P and Q $(P;Q)$. The CC value both ladder diagram P and Q are $1 * (2 + 1) = 3$. The summation value of CC for P and Q ladder diagram is 6 which is less than the CC vale of $P;Q(CC(P;Q) = (1 + 2) * (3 + 1) = 12))$. The example demonstrates that the CC value of two concatenating program is greater than the summation of their CC values. Hence, CC metric satisfies Property 9.

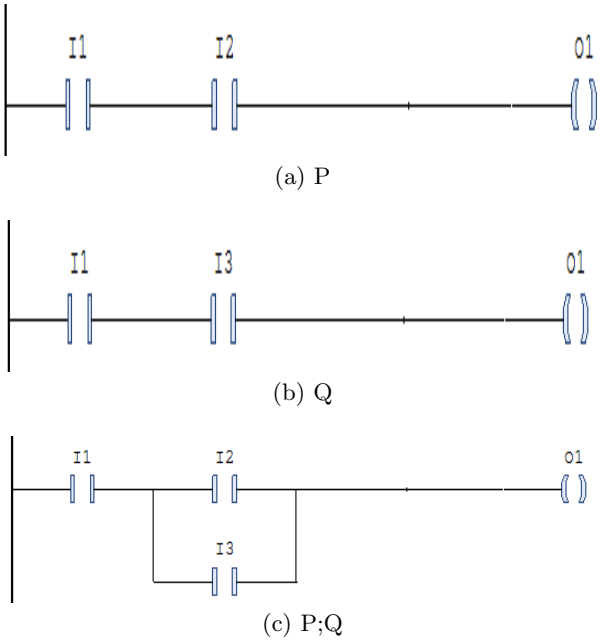


Figure 7: Examples of two sequential PLC Ladder Diagram and their concatenation

Table 2 the Weyuker's property analysis for all the proposed metric. We present the detailed analysis of Weyuker's property in this paper only for the CC and not for other metrics due to the limited space in the paper. Table 2 reveals that the CC metric satisfies 7 out of the 9 properties. The TC metric also satisfies 7 properties. Analysis of Weyuker's properties for software metrics is an area which has attracted several researcher's attention. Previous research shows [8] that satisfying all the 9 properties is not a necessary condition for a metric to be adopted. Satisfying various Weyuker's properties is a validation step which

Table 2: Weyuker Properties and Source Code Metrics. The symbol (denoting yes or no) in Each Cell shows if the Property is Satisfied or Not

Properties	Size	VC	CC	TC
Property 1	✓	✓	✓	✓
Property 2	✓	✓	×	×
Property 3	✓	✓	✓	✓
Property 4	✓	✓	✓	✓
Property 5	✓	×	✓	✓
Property 6	✓	×	✓	✓
Property 7	×	×	✓	✓
Property 8	✓	✓	✓	✓
Property 9	×	×	×	×

Table 3: The Value for all 4 Metrics for the Worked-Out Examples and Illustrative Figures

Example	Size	VC	CC	TC
Figure 1a	4	6	3	1
Figure 1b	5	7	9	3
Figure 1c	13	15	49	6
Figure 2a	16	10	56	15
Figure 2b	16	10	120	15
Figure 3a	4	6	4	1
Figure 3b	4	6	12	3
Figure 4a	3	8	15	1
Figure 4b	6	8	15	3
Figure 5a	3	5	3	1
Figure 5b	5	8	15	2
Figure 6a	3	5	3	1
Figure 6b	3	5	3	1
Figure 6c	3	5	3	1
Figure 6d	4	6	12	3
Figure 6e	6	7	10	2
Figure 7a	3	5	3	1
Figure 7b	3	5	3	1
Figure 7c	4	6	12	3

indicates the strength or quality of the metric but satisfying all the properties is not mandatory. We observe that in the original work of Weyuker's [12], the author did not find a single complexity measures (statement count, cyclomatic number, effort measure, data flow complexity) which is satisfied by all 9 properties.

Table 3 shows the calculated values for the 4 proposed metrics for the worked-out examples and the ladder diagrams in various Figures presented in the paper. Table 3 reveals that the size, cognitive complexity and testing complexity captures different aspects of a program. We observe that the gap between cognitive and testing complexity various significantly for different ladder diagrams depending on the structure and the control structures used.

5. CONCLUSION

We propose vocabulary, size, cognitive complexity and testing complexity metrics for a visual programmable control logic programming language. The proposed metric for ladder diagram which is part of the IEC 61131-3 standard are defined keeping in mind the software measurement application such as cost and effort estimation, measuring productivity of programmers and measuring the quality of the product. The proposed metric is based on various factors such as the number of distinct operators and operands, the total number of operators and operands, cognitive weights of various control structures and the overall control flow of the program. We apply Weyuker's properties on the proposed metrics and present the list of properties which are satisfied and the ones which are not satisfied. We present examples to demonstrate that two programs having the same size can have different cognitive and testing complexity. Similarly, the cognitive and testing complexity of programs can vary significantly depending on the structure and the basic elements used.

6. REFERENCES

[1] Cardoso, J., Mendling, J., Neumann, G., and Reijers, H. A. A discourse on complexity of process models. In *Business Process Management Workshops* (2006), BPM'06, pp. 117–128.

[2] Fenton, N., and Bieman, J. *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[3] Gharieb, W. Software quality in ladder programming. In *Computer Engineering and Systems, The 2006 International Conference on* (2006), pp. 150–154.

[4] Gruhn, V., and Laue, R. Complexity metrics for business process models. In *in: W. Abramowicz, H.C. Mayr (Eds.), Business Information Systems, Lecture Notes in Informatics* (2006), pp. 1–12.

[5] John, K.-H., and Tiegelkamp, M. *IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids*. Springer Science & Business Media, 2010.

[6] Lanza, M., and Marinescu, R. *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media, 2007.

[7] Lucas, M., and Tilbury, D. Methods of measuring the size and complexity of plc programs in different logic control design methodologies. *The International Journal of Advanced Manufacturing Technology 26*, 5 (9 2005), 436–447.

[8] Misra, S., and Akman, I. Applicability of weyuker's properties on oo metrics: Some misunderstandings. *Computer Science and Information Systems 5*, 1 (2008), 17–23.

[9] Nair, A. Product metrics for iec 61131-3 languages. In *Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on* (2012), pp. 1–8.

[10] Otto, A., and Hellmann, K. Iec 61131: A general overview and emerging trends. *Industrial Electronics Magazine, IEEE 3*, 4 (2009), 27–31.

[11] Waters, M., Young, K., and Baxter, I. D. Calculating software metrics for ladder logic. In *ICINCO-RA (2)* (2008), J. Filipe, J. Andrade-Cetto, and J.-L. Ferrier, Eds., pp. 143–150.

[12] Weyuker, E. Evaluating software complexity measures. *Software Engineering, IEEE Transactions on 14*, 9 (1988), 1357–1365.