Comparison of Code Measures of IEC 61131-3 and 61499 standards for Typical Automation Applications

Peter Gsellmann
Automation and Control Institute
TU Wien
Vienna, Austria
gsellmann@acin.tuwien.ac.at

Martin Melik-Merkumians
Automation and Control Institute
TU Wien
Vienna, Austria
melik-merkumians@acin.tuwien.ac.at

Georg Schitter
Automation and Control Institute
TU Wien
Vienna, Austria
schitter@acin.tuwien.ac.at

***Abstract*—The IEC 61131-3 and IEC 61499 standards are currently used to implement automation systems. Although aspects as reuseability, execution models, and component-based design have been discussed and analyzed for both standards, the programming effort and complexity of implementation for typical tasks in industrial automation have not been compared so far. This work proposes two typical application classes of industrial automation systems and implements one concrete application for each class with IEC 61131-3 and IEC 61499, respectively. Both implementations are then evaluated via McCabe's Cyclomatic Complexity and Halstead software measures, providing an objective comparison on the implementations.**

***Index Terms*—IEC 61131, IEC 61499, software metric, comparison**

I. Introduction

Control software is a central component in today's automation systems and its complexity is continuously increasing [1]. Industry developed two different standards to cope with software development in industrial automation, the IEC 61131-3 – Programmable controllers: Programming languages [2] for centralized, cyclic Programmable Logic Controller (PLC) systems, and the IEC 61499 – Function blocks [3] for distributed, event-driven automation systems. The IEC 61499 was originally devised for modeling distributed systems and coordination between classic IEC 61131-3 driven PLC devices [4], but the first reference implementations of the IEC 61499, FBDK [5] and Eclipse 4diac [6], instead implemented full development and run-time environment for automation systems without the need for IEC 61131-3 devices. This enabled the development of complete automation solutions within the IEC 61499 standard. As both standards can be used to implement automation control code, the advantages and disadvantages of IEC 61499 and IEC 61131-3 regarding the aspects of reuseability, execution models, and component-based design [7–10] have been evaluated and discussed, but so far the comparison of programming complexity and programming effort for typical automation applications has been neglected. The aim of this work is to evaluate the complexity and programming effort of implementations in IEC 61131-3 Function Block Diagram (FBD) and Structured Text (ST), and IEC 61499 in order to give first results which programming model is better qualified for certain kinds of typical automation tasks.

II. Related Work

The application of software metrics is an established method of evaluating code complexity, quality, and effort via certain code characteristics [11]. For the evaluation on the suitability of the different standards, measures are needed which quantify the general complexity of the program structure, in order to be sure that both programs have the same number of decision points. This is important for a valid comparison, as programs which provide more functionality, will tend to be larger and more complex than programs with less functionality. Only if both implementations can be considered equal in terms of implemented functionality, a metric can be applied to compare programming effort in order to identify the more eligible language.

A. Software Metrics

A software metric is a function, with software data as input quantity, and a numerical value as the output [12]. The resulting assessment value gives a quantitative evaluation of the examined program. Since it is not possible to define a reference value, and subsequently introduce units, software metrics only serve to compare applications against each other.

One of the oldest and widely used metrics is Lines of Code (LOC), and was initially used to measure the productivity of programmers. The calculation is quite simple, as the metric's resulting value is the aggregate of the lines of the source code. One major drawback of this metric is, that the computational effort for a line of code is not considered in the metric, which hampers its ability to give an accurate measure of software complexity. Another problem is, that it cannot be directly applied to graphical programming languages [13], such as IEC 61131-3 FBD.

A metric which also takes the complexity of program statements into account are the Halstead complexity measures [14], defining a class of metrics, that rely on the classification

Table I
HALSTEAD SOFTWARE MEASURES AND CALCULATION FORMULAS

Metric	Calculation formula
Program vocabulary	$n = n_1 + n_2$
Program length	$N = N_1 + N_2$
Estimated length	$\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$
Purity ratio	$PR = \frac{\hat{N}}{N}$
Program volume	$V = N \cdot \log_2 n$
Program difficulty	$D = \frac{n_1}{2} \cdot \frac{N_2}{n_2}$
Program effort	$E = D \cdot V$

of program tokens as *operators* or *operands*. The software measures (cf. Table I) of this metric depend on the count of these tokens, as defined in Eqs. (1a) - (1d).

$$n_1 = \text{number of unique operators} \tag{1a}$$
$$n_2 = \text{number of unique operands} \tag{1b}$$
$$N_1 = \text{total number of operators} \tag{1c}$$
$$N_2 = \text{total number of operands} \tag{1d}$$

The goal of the Halstead complexity measures is to give estimates on the difficulty to understand or write a program (D) and the effort to write the program (E).

Another widely used software metric is McCabe's Cyclomatic Complexity (CC) [15]. The CC indicates the number of linearly independent paths in the control flow. McCabe's CC hereby counts the number of decisions d in a program, or respectively the number of edges e and nodes n in a graphical representation, shown in Eq. (2), as a measure of program complexity.

$$CC = d + 1 = e - n + 2 \tag{2}$$

B. Software metrics adapted to IEC 61131-3

With a view to use software metrics as an indicator for Technical Debt (TD), [13] published an adaption of the code metrics, presented in Section II-A, to IEC 61131-3 applications.

Since Halstead's metric relies on the unique and total number of *operators* and *operands*, these tokens have to be related to the elements of a FBD. According to [16], the relation is as follows: All connections count as *operators* while all used Function Blocks (FBs) count as *operands*. In contrast, the metric can be directly applied to ST, since the tokens are apparent.

For the elicitation of the CC number, [13] used the coherence from Eq. (2). The application of the metric on ST is again direct, as only the conditional statements, such as `IF-THEN-ELSE` or `CASE-OF`, have to be counted. The method for FBD figures similar, because the IEC 61131-3 defines FBs, which represent conditional statements, such as *SEL* and *MUX* [17].

C. Software Metrics Adapted to IEC 61499

A set of rules to obtain the number of *operators* and *operands* for IEC 61499 applications was presented in [18].

For algorithms implemented in ST, which are combinations of valid *operators* and *operands*, the metric can be applied directly. For the evaluation of the Execution Control Chart (ECC) the metric needs to be adapted. An ECC state with its transitions is comparable to a decision point, where, depending on the transition, the program flow proceeds to a new state. Therefore, a combination of a state and a transition is counted as one *operator*. Additionally, an action, consisting of an output event and/or an algorithm, is considered as an *operator* too. *Operators* act upon *operands*. This means, transitions, as well as algorithms and output events of actions assigned to states, are considered as *operands*.

The determination of the CC measure for IEC 61499 ECCs is an adaptation of the general rules for evaluating graphical program representations. All states are hereby considered as *nodes* and all transitions are considered as *edges*, enabling the calculation of CC via Eq. (2). For algorithms in ST, the metric can be again applied direct, as only the conditional statements have to be counted. According to [18], the combination of algorithm and ECC measures gives an idea of the overall complexity, but does not indicate where it is hidden. For the present analysis, the location of the complexity is negligible.

III. EXAMPLE APPLICATION

In order to evaluate the languages of IEC 61131-3 and IEC 61499 in respect to their suitability for the implementation of typical application classes in industrial automation, a typical flow control application and a cyclic application have been chosen as first evaluation scenarios. The implementations of the two applications are compared in respect to their amount of linearly independent application paths via McCabe's CC and their understandability and effort of implementation via Halstead complexity measures. It is expected that the implementations in both standards exhibit a similar CC, as both implementation shall perform the same task. If the CC is similar or equal, the Halstead complexity measures can give comparable estimates for program understandability and programming effort, although we compare different languages [14]. For all IEC 61131-3 implementations, the periodic execution behavior, as defined in [19], is used. All IEC 61499 Basic Function Block (BFB) algorithms are programmed in ST.

The first example application is a pick-and-place task for a resistor sorting plant, shown in Fig. 1. It is one instance of the application class of sequencer applications, a typical task in the industrial automation. The task is defined as follows:

- The vibrator unit shakes a separated resistance piece to the outlet, from where the portal robot can pick it up.
- Afterwards, the resistance piece is placed in the measurement station, where its resistance value is measured.
- If the resistance is within or outside a certain tolerance range, the piece gets carried to the first or second palette, respectively.

A second typical application class is the control of process variables, hence the second sample application is the tank level control of a process tank system, shown in Fig. 2. In this example a Proportional-Integral (PI) controller controls the level of upper tank *T102* to a certain level, via the pump *P101*, while the liquid runs off into the lower tank *T101* via valve *V102*.

IV. Results and Discussion

For the resistor sorting application in IEC 61499, three FBs are developed, a sequence control FB, a movement control FB and a measurement station control FB, while for the case of IEC 61131-3, only one FB with an internal ST algorithm is designed. Table II shows the resulting software measures for the resistor sorter application, implemented in the standards IEC 61131-3 and IEC 61499. The complexity for the IEC 61131-3 application is, against expectation, lower than for the IEC 61499 application. The reason for the higher complexity are the additional `CYCLE` FBs, used for the periodic scan of the sensors. If the complexity added by these FBs, which add nothing to the programs base functionality, is ignored, the CCs for both applications are equal. The measures for program length, program vocabulary, program difficulty, and program effort are significantly higher from the IEC 61131-3 than from the IEC 61499 application. Considering that the IEC 61131-3 program is implemented in FBD and ST, and not in the likely more suitable Sequential Function Chart (SFC), these results are comprehensible.

In the second test case, the tank level control, both, the IEC 61499 and the IEC 61131-3 application consist of one FB, which has the same PI control algorithm implemented. Table III shows the resulting software metric values for the tank level controller, implemented in the standards IEC 61131-3 and IEC 61499. For the implementation in IEC 61499, the Halstead complexity measures are significantly higher. This is due to the fact that in IEC 61131-3, the controlling algorithm implementation is just one calculation, whereas in a IEC 61499 FB, it also needs an ECC. Again, with the neglection of the `CYCLE` FB the CC number is the same.

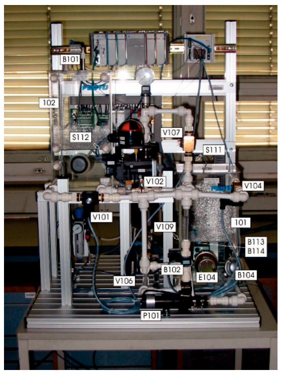

Figure 2. Example application: Tank level control

The results show, that IEC 61131-3 and IEC 61499 languages have their pros and cons, based on the application type. This is partly due to the language, partly based on the underlying execution semantics. As expected for the used IEC 61131-3 languages, the IEC 61499 is better suited for sequencer applications, whereas IEC 61131-3 is preferably used for periodic controller applications.

V. Conclusion & Future Work

This work proposes two typical classes of applications in industrial automation, compares two concrete equivalent implementations, one for each standard, and discusses the suitability of the standards for these application classes via software metrics. In the example of the resistor sorting process, the CC number is equal for both programs indicating functional equivalent implementations, whereas the Halstead software measures, such as program length and program effort, are significantly smaller in the case of the IEC 61499 application. Therefore, according to the software measure results, the IEC 61499 is more suitable for sequencer applications, than equivalent IEC 61131-3 FBD/ST applications. For the class of control applications, represented in the tank level control application, the complexity is again equivalent, indicating that both implementations are functionally equivalent. The

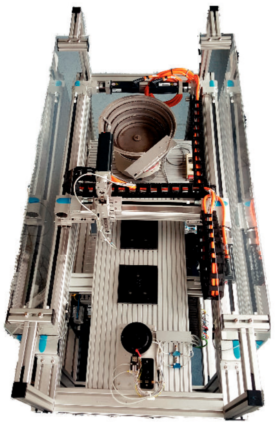

Figure 1. Example application: Resistor sorter

Table II
Metric Results of Resistor Sorting Application for IEC 61131-3 and IEC 61499. For the IEC 61499 implementation, the cyclomatic complexity value given in parentheses excludes the added complexity via the extraneous CYCLE FBs.

	IEC 61131-3	**IEC 61499**
Program vocabulary	587	217
Program length	581	439
Estimated length	4839.35	1503.98
Purity ratio	8.33	3.43
Program volume	5343.58	3407.32
Program difficulty	109.50	46.49
Program effort	585122.33	158408.48
Cyclomatic Complexity	33	45(33)

Table III
METRIC RESULTS OF TANK LEVEL CONTROLLING APPLICATION FOR IEC 61131-3 AND IEC 61499. FOR THE IEC 61499 IMPLEMENTATION, THE CYCLOMATIC COMPLEXITY VALUE GIVEN IN PARENTHESES EXCLUDES THE ADDED COMPLEXITY VIA THE EXTRANEOUS CYCLE FBS.

	IEC 61131-3	**IEC 61499**
Program vocabulary	68	82
Program length	61	117
Estimated length	346.63	455.60
Purity ratio	5.68	3.89
Program volume	371.34	743.83
Program difficulty	15	11.77
Program effort	5570.03	8758.04
Cyclomatic Complexity	3	6(3)

evaluation of the IEC 61131-3 application shows smaller measures for most of the Halstead software measures, except program difficulty, which is slightly higher than, but still in the same range as the IEC 61499 implementation. From this, it can be concluded, that the IEC 61131-3 is better suited for control algorithm implementations.

The next step in the evaluation will be the incorporation of the missing IEC 61131-3 languages SFC and Ladder Diagram (LD). Additionally, also the effect of the object-oriented extension of the IEC 61131-3 standard has to be evaluated. Furthermore, for very simple FB implementations, such as the PI FB from the tank level control application, the impact of replacing the BFB implementation, with a Simple FB is of interest, due to the decreased complexity as no ECC is needed for this type of FB.

REFERENCES

[1] W. ElMaraghy, H. ElMaraghy, T. Tomiyama, and L. Monostori, “Complexity in engineering design and manufacturing”, *CIRP Annals*, vol. 61, no. 2, pp. 793–814, 2012.

[2] International Electrotechnical Commission, *IEC 61131 – Programmable controllers, Part 3: Programming languages*, 2013.

[3] IEC TC65/WG6, *IEC 61499: Function blocks for industrial-process measurement and control systems – Parts 1 to 4*. Geneva: International Electrotechnical Commission (IEC), 2005.

[4] R. Schoop and A. Strelzoft, “Asynchronous and synchronous approahces for programming distributed control systems based on standards”, *Control Engineering Practice*, vol. 4, no. 6, pp. 855–861, 1996.

[5] I. HOLOBLOC. (May 29, 2018). Fbdk 2.6 - the function block development kit, [Online]. Available: http://www.holobloc.com/fbdk2/.

[6] Eclipse. (2016). 4diac - Framework for Industrial Automation & Control, [Online]. Available: https://eclipse.org/4diac/.

[7] A. Zoitl and V. Vyatkin, “Different perspectives [face to face; "IEC 61499 architecture for distributed automation: The ‘"glass half full" view]”, *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 7–23, 2009.

[8] L. A. C. Salazar and O. A. R. Alvarado, “The future of industrial automation and IEC 614993 standard”, in *2014 III International Congress of Engineering Mechatronics and Automation (CIIMA)*, Oct. 2014, pp. 1–5.

[9] K. Thramboulidis, “Different perspectives [face to face; "IEC 61499 function block model: Facts and fallacies"]”, *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 7–26, Dec. 2009.

[10] ——, “IEC 61499: Back to the well proven practice of IEC 61131?”, in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, Sep. 2012, pp. 1–8.

[11] R. Lincke, J. Lundberg, and W. Löwe, “Comparing software metrics tools”, in *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ser. ISSTA ’08, Seattle, WA, USA: ACM, 2008, pp. 131–142.

[12] T. Honglei, S. Wei, and Z. Yanan, “The research on software metrics and software complexity metrics”, in *2009 International Forum on Computer Science-Technology and Applications*, vol. 1, Dec. 2009, pp. 131–136.

[13] L. Capitán and B. Vogel-Heuser, “Metrics for software quality in automated production systems as an indicator for technical debt”, in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, Aug. 2017, pp. 709–716.

[14] M. H. Halstead, “Elements of software science”, *Elsevier New York*, 1977.

[15] T. J. McCabe, “A complexity measure”, in *Proceedings of the 2Nd International Conference on Software Engineering*, ser. ICSE ’76, San Francisco, California, USA: IEEE Computer Society Press, 1976, pp. 407–.

[16] A. Muslija, “On the complexity measurement of industrial control software”, Master Thesis, Mälardalen University, School of Innovation Design and Engineering, Västeras, Sweden, 2017.

[17] K. H. John and M. Tiegelkamp, *IEC 61131-3: Programming Industrial Automation Systems Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*, 2nd. Springer Publishing Company, Incorporated, 2010.

[18] G. Zhabelova and V. Vyatkin, “Towards software metrics for evaluating quality of IEC 61499 automation software”, in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sep. 2015, pp. 1–8.

[19] International Electrotechnical Commission, *IEC TR 61131 – Industrial-process measurement and control - Programmable controllers, Part 8: Guidelines for the application and implementation of programming languages*, 2017.