A Software Measure for IEC 61499 Basic Function Blocks

Lisa Sonnleithner, Alois Zoitl, *Member, IEEE*
LIT CPS Lab, Johannes Kepler University Linz
Linz, Austria
{lisa.sonnleithner, alois.zoitl}@jku.at

***Abstract*—When proposing new design patterns or design guidelines for IEC 61499 Applications, their performance and quality has to be compared to alternative solutions. Software measures are a way of achieving that. Although first attempts towards software measures for IEC 61499 were made, the standard still lacks a solid set of measures to evaluate applications. Taking a first step in that direction, we propose a set of measures for an IEC 61499 Basic Function Block.**

***Index Terms*—IEC 61499, distributed control system, software measure, software metric**

I. Introduction

In recent years, the share of software in modern industrial systems is increasing significantly [1], [2]. Not only the share of software is increasing, but also its complexity [1], [3]. Hence, the need for design guidelines arises. But how do we evaluate the quality of such design guidelines?

Software measurement has become an essential part of software engineering. Measurement in general means assigning a value to an attribute of an entity to compare entities. Therefore, software measurement aims to quantify certain attributes of a software product. In the last decades, many software measures and metrics[1] were developed. There exists a number of generic measures that claim to be applicable to any programming language and various measures for a certain programming language. Unfortunately, in the domain of component–based design, software measures are scarce [5].

In [6], the authors made a first step towards defining software metrics for IEC 61131 [7]. The authors adapted various existing metrics and evaluated the results. The metrics were validated and their usefulness was graded by practitioners. Software metrics specific to IEC 61499 [8], on the other hand, have yet to be investigated. In [9], an initial attempt towards software metrics in IEC 61499 was made. The authors also applied existing metrics, including Halstead's metrics [10] and McCabe's cyclomatic complexity [11], on Function Blocks (FBs). The metrics of all FBs of an application are averaged to obtain the metrics of that application. Unfortunately, this approach leads to some shortcomings that we discovered in a previous work [12].

There we compared three different design patterns for distributed control applications. Based on an example system, three different control applications were developed in 4diac IDE [13]. Thereafter, the system was modified by expanding the process in two different ways. The control applications were adapted accordingly. Finally, the applications and their extensions were compared, using the software metrics for IEC 61499 proposed in [9]. As anticipated, the result was in favor of a distributed design. What we did not expect was, that the metrics for one adaptation decreased compared to the original system. The reason for this effect was that the metrics of all FBs in the application are averaged to generate the application metric. That points towards the need of adjustments to IEC 61499 metrics.

To overcome the identified limitations, this paper investigates software measurement in general and how to apply it to IEC 61499 Basic Function Blocks (BFBs) for getting better feedback about the FB's design.

In Section II, we will look into existing metrics and software measurement in general. Based on the findings of that Section, we will then propose our first step towards profound IEC 61499 measures in Section III. In Section IV, we will pre-evaluate our measures and how they perform compared to other measures. Finally, the paper is concluded in Section V.

II. Background

Measuring attributes of interest plays a key role in most engineering disciplines. In software engineering, however, measurements often lack a solid theoretical basis [14]. Even well–established software metrics, such as Halstead's suite [10] or McCabe's cyclomatic complexity [11], have been criticized [15]–[18]. On the other hand, newly developed software measures and metrics are either not validated at all or validated by showing a correlation with existing ones. This holds not only the pitfall of correlating against a non validated measure or metric, but also of finding a spurious correlation [15], [19]. To avoid these issues, a closer look into the discipline of measurement and how to extend it to the domain of software engineering is needed.

[1]Measures refer to concrete attributes, e.g., Lines of Code, whereas metrics refer to abstract attributes, e.g., quality [4].

Several authors already tackled this issue [15], [20], [21]. The authors of [20] proposed that a measurement is valid, if the following points are valid:

- The **Attribute** we are interested in is valid, if it is exhibited by the entity we are measuring.
- The measurement **Unit** is valid, if it is an appropriate means to measure the attribute.
- The measurement **Instrument** is valid, if it's underlying model is valid and if it is properly calibrated.
- The measurement **Protocol** is valid, if it lets us measure an attribute consistently and repeatably.

Another way to validate software measures was proposed in [22] and in a similar way in [23]. The authors propose to first define a set of properties, that a measure needs to fulfill. A measure that does not fulfill one property is not a valid measure that fulfills all but one property, but rather is no valid measure at all [23].

The authors in [23] propose that a size measure of a system S with modules $m_i \subseteq S$ needs to fulfill the following properties:

- **Nonnegativity**
  The size of a system can not be negative:
$$\text{Size}(S) \geq 0 \tag{1}$$
- **Null Value**
  The size of an empty system is zero
$$\text{Size}(S = 0) = 0 \tag{2}$$
- **Module Additivity**
  The size of a system is equal to the sum of the size of its disjoint modules.
$$\begin{aligned}&(m_1 \cap m_2 = S \text{ and } m_1 \cup m_2 = 0)\\ &\Rightarrow \text{Size}(S) = \text{Size}(m_1) + \text{Size}(m_2)\end{aligned} \tag{3}$$
- **Size monotonicity**
  Adding modules does not decrease size.
$$\begin{aligned}&(m_1 \cup m_2 = S \text{ and } m_1 \cup m_2 \neq 0)\\ &\Rightarrow \text{Size}(S) \leq \text{Size}(m_1) + \text{Size}(m_2) \text{ and}\\ &\quad \text{Size}(S) \geq \max(\text{Size}(m_1), \text{Size}(m_2))\end{aligned} \tag{4}$$

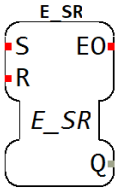
The properties proposed in [22] are more restrictive. One property, for example, requires the measure to not be too coarse, meaning it is undesirable for a measure to lead to the same results for too many different systems. As it would exceed the scope of this paper, we will not discuss all properties in detail, please refer to the original work for reference.

III. Proposed Software Measurement for IEC 61499 BFBs

Utilizing this theoretical background, we will propose a set of measures that measure the direct attributes of a BFB. In general, the attributes of an entity can be divided in direct and derived attributes. A direct attribute can be measured directly, e.g., the height of an object, whereas a derived attribute, e.g., the density of an object, has to be calculated from other direct attributes [15]. This implies that a clear understanding of the relationship between attributes is needed to calculate derived attributes. Especially in software engineering, we lack this clear understanding, two software engineers might even argue about the definition of an attribute itself, e.g., complexity. This makes the attempt of finding a metric of a complex derived attribute doomed to fail. Therefore, we will focus on direct attributes, that will be validated according to the properties summarized in Section II. Before we introduce our proposed measures, we will shortly explain IEC 61499 BFBs, the atomic units of IEC 61499 Control Applications.

A. IEC 61499 BFBs

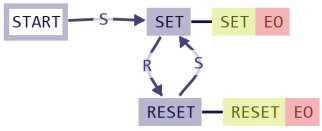
An IEC 61499 BFB consists of an interface and an Execution Control Chart (ECC), see Fig. 1a and Fig. 1c respectively. The interface shows all event and data connections leading to and from the BFB. Inside, the BFBs functionality is modeled by an ECC. The ECC is a state diagram that is responsible for controlling the execution of actions. It consists of states (blue) that are connected by transitions. Each transition may also have a condition, e.g., a certain input event arriving. When entering a new state, it's algorithms (green) are executed and output events (red) are fired, also called actions. Afterwards, the current state stays active until an outgoing transition condition is fulfilled and the next state is entered. Additionally, an ECC may have internal variables. Fig. 1b and Fig. 1d summarize the attributes of a BFB interface and ECC.



(a) Interface of a BFB.

Interface
Input Events
Output Events
Input Data
Output Data
Adapters

(b) Attributes of a BFB Interface.



(c) ECC of a BFB.

ECC
States
Transitions
Algorithms
Actions
Internal variables

(d) Attributes of a BFB ECC.

Fig. 1: IEC 61499 BFB.

B. Newly proposed measures

Instead of combining these attributes, attempting to come up with a revolutionary metric that may or may not fulfill the properties of Section II, each of them will be part of a measure set. Our measure set will therefore consist of size measures of each attribute (e.g., the total count of all states). Additionally to the BFBs attributes listed in the previous section, we include the ECC's independent paths. McCabe's Cyclomatic complexity has been criticized as a complexity

measure, but it is a valid measure for independent paths of a graph. As the ECC is already a graph, it seems obvious to include that attribute. To measure the size of the algorithms we use Lines of Code (LoC). A line of code will be counted, if it is not a blank line or a comment.

Instead of choosing a table to represent this measure set, we propose to plot the measures in a spider chart. We call these new measures *Spider Chart BFB Measures*. The *Spider Chart BFB Measures* for the FB from Fig. 1 are shown in Fig. 2. The proposed *Spider Chart BFB Measures* provide the following advantages:

- They are easy to compute.
- They are easy to validate (cf. Section II), ensuring a sound base for future work.
- Spider charts facilitate comprehensibility compared to tables.
- Calculation and visualization of *Spider Chart BFB Measures* can easily be implemented in any IEC 61499 development environment.

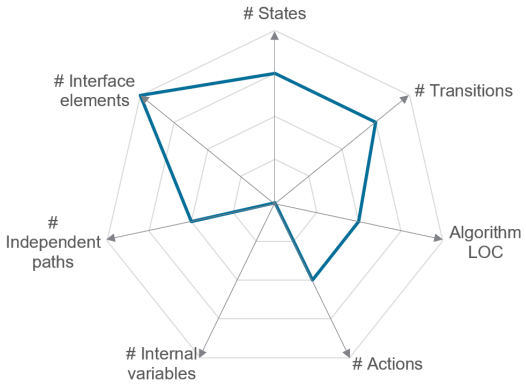


Fig. 2: *Spider Chart BFB Measures* of the example BFB shown in Fig. 1.

C. Validation of Proposed Metric

To validate the *Spider Chart BFB Measures*, we will check whether the properties introduced in Section II are fulfilled. We will start with the properties proposed by [20].

- All **Attributes** included in the *Spider Chart BFB Measures* can be directly associated with a BFB.
- The **Units** of all attributes are a count based on a reference element.
- The **Instrument** to measure the attributes will be a program that performs the counting of each attribute.
- The measurement **Protocol**, e.g., counting all states of a BFB, has to be implemented correctly in the program.

Therefore, the measurement of all attributes is valid according to [20]. Next, we will check the properties suggested by [23].

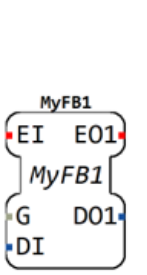
- **Nonnegativity**: This property is fulfilled by all sub-measures of the *Spider Chart BFB Measures*.
- **Null Value**: This property is fulfilled by all sub-measures of the *Spider Chart BFB Measures*.
- **Module Additivity**: Interface elements, states, transitions, internal variables and independent paths of the ECC are unique, therefore, e.g., adding one additional state will always increase the state count by one. An algorithm on the other hand, can be called by several states. Calling an existing algorithm a second time will not increase algorithm LoC. Adding a new algorithm will increase algorithm LoC by the lines of code of that algorithm. Therefore, algorithm LoC fulfills this property.
- **Size Monotonicity**: The same line of argumentation that was used for Module Additivity can be used here and therefore, all sub-measures fulfill Size Monotonicity.

The *Spider Chart BFB Measures* fulfill all size properties, as introduced in [23]. The successful validation of the *Spider Chart BFB Measures* provides a solid basis to build upon. In the next section the *Spider Chart BFB Measures* will be applied to two example BFBs to get a first impression of their practicability. Additionally, they will be compared to previous IEC 61499 measures.

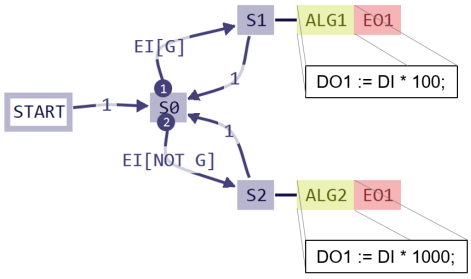
IV. Pre-evaluation

As an example, two different implementations of the same functionality are compared. The example functionality is, that depending on whether the Boolean input G is set to $true$ or $false$, the data input DI is multiplied by a different constant. The result is stored in the data output $DO1$. After setting the output, an output event $EO1$ is fired.

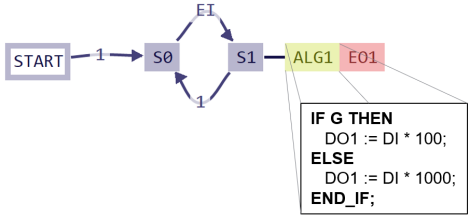
When designing a BFB, there are several solutions that lead to the same functionality. The first version, $MyFB1$, implements most of the functionality as states and appropriate transition conditions, as shown in Fig. 3b. In contrast to that, the second version, $MyFB2$, implements most of the functionality inside of an algorithm, as shown in Fig. 3c. Both BFBs have the same interface, which is shown in Fig. 3a. Experts confirm that the way this functionality is implemented strongly depends on the personal preferences of the automation engineer. Furthermore, two FBs with the same functionality



(a) Interface of $MyFB1$ and $MyFB2$. Events are marked in red, Boolean inputs in grey and integer in and outputs in blue.



(b) ECC of $MyFB1$.



(c) ECC of $MyFB2$.

Fig. 3: Implementation of $MyFB1$ and $MyFB2$.

are compared. This suggests that the results for both FBs should be within a similar range.

Fig. 4 shows that the spider chart of $MyFB1$ has high values in States and Transitions. Compared to that, $MyFB2$ shows a high value in Algorithm LoC. The measures indicate that two FBs of similar scope are compared.

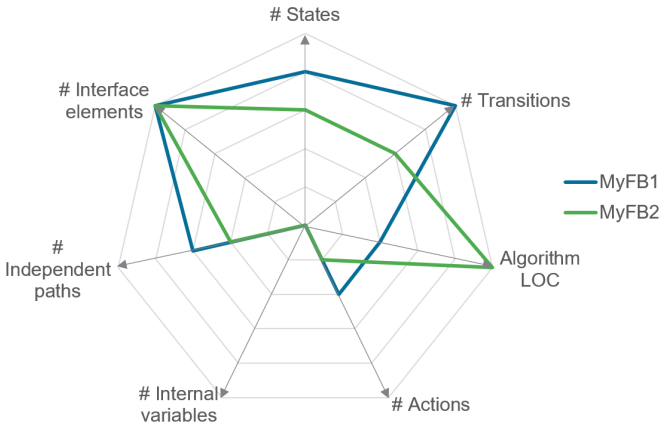


Fig. 4: *Spider Chart BFB Measures* of $MyFB1$, shown in blue, and $MyFB2$, shown in green.

In contrast to that, the metrics proposed in [9] show different results, see Table I. The suggested Program Effort of $MyFB1$ is more than twice the Program Effort of $MyFB2$. By comparing these results, we can clearly see the advantages of the proposed *Spider Chart BFB Measures*. The measures of the two BFBs are in a similar range and we can immediately see where the BFB's functionality is mainly implemented. Therefore, the actual rating of a BFB not only depends on the values of the measures, but also on the subjective preferences of the automation engineer. In contrast to that, existing metrics put ECC-predominant implementations on a disadvantage. A drawback of the proposed *Spider Chart BFB Measures* might be that this dependency on the preferences of an automation engineer complicates automatic comparison of BFBs.

V. Conclusion

Software metrics are an important means to compare implementations and to assess software quality. After providing a solid theoretical background about software measurement, we proposed a new set of software measures for IEC 61499. The proposed *Spider Chart BFB Measures* are promising, which is not only shown by the example in Section IV, but also by comparison to previous measures. As the *Spider Chart BFB Measures* were validated successfully they provide a solid basis to build upon.

In future work, we will investigate critical thresholds that aim at indicating when a single BFB implements too much functionality and should therefore be split. This approach is already state of the art in development environments of many programming languages.

Most importantly, we will focus on how to extend the proposed measures to Composite FBs and to IEC 61499 Applications. This should not only provide a way for scientists to evaluate their potential new design guidelines, but also support automation engineers in designing applications.

TABLE I: To IEC 61499 adapted Halstead's metrics as proposed in [9].

Metrics	**MyFB1**	**MyFB2**
Program length	28	21
Program vocabulary	21	17
Estimated length	73	55
Program volume	123	86
Difficulty	4.25	2.71
Program effort	523	232

Acknowledgements

This work has received funding from the European Union's 1-SWARM project under grant agreement 871743.

References

[1] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234–1249, 2013.

[2] B. Vogel-Heuser, C. Diedrich, A. Fay, S. Jeschke *et al.*, "Challenges for software engineering in automation," *Journal of Software Engineering and Applications*, vol. 07, no. 05, pp. 440–451, 2014.

[3] M. Törngren and P. Grogan, "How to deal with the complexity of future cyber-physical systems?" *Designs*, vol. 2, no. 4, p. 40, 2018.

[4] "Metrics and measures," https://samate.nist.gov/index.php/Metrics_and_Measures.html, accessed: 2020-06-12.

[5] T. Vale, I. Crnkovic, E. S. De Almeida, P. A. D. M. S. Neto *et al.*, "Twenty-eight years of component-based software engineering," *Journal of Systems and Software*, vol. 111, pp. 128–148, 2016.

[6] J. Wilch, J. Fischer, E.-M. Neumann, S. Diehm *et al.*, "Introduction and evaluation of complexity metrics for network-based, graphical iec 61131-3 programming languages," in *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1. IEEE, 2019, pp. 417–423.

[7] IEC, "IEC 61131 - programmable controllers, part 3: Programming languages," Geneva, 2013. [Online]. Available: www.iec.ch

[8] IEC TC65/WG6, "IEC 61499-1, function blocks - part 1: architecture v2.0: Edition 2.0," Geneva. [Online]. Available: www.iec.ch

[9] 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)*, 2015.

[10] M. H. Halstead, "Elements of software science," 1977.

[11] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[12] B. Wiesmayr, L. Sonnleithner, and A. Zoitl, "Structuring distributed control applications for adaptability," in *3rd IEEE conference on industrial cyber-physical systems*, 2020 in press.

[13] Eclipse 4diac, "Eclipse 4diac - the open source environment for distributed industrial automation and control systems," 2020. [Online]. Available: https://www.eclipse.org/4diac

[14] N. Fenton, "Software measurement: A necessary scientific basis," *IEEE Transactions on software engineering*, vol. 20, no. 3, pp. 199–206, 1994.

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

[16] P. G. Hamer and G. D. Frewin, "M.h. halstead's software science - a critical examination," in *Proceedings of the 6th International Conference on Software Engineering*, ser. ICSE '82. Washington, DC, USA: IEEE Computer Society Press, 1982, p. 197–206.

[17] V. Y. Shen, S. D. Conte, and H. E. Dunsmore, "Software science revisited: A critical analysis of the theory and its empirical support," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 2, pp. 155–165, 1983.

[18] M. Shepperd, "A critique of cyclomatic complexity as a software metric," *Software Engineering Journal*, vol. 3, no. 2, pp. 30–36, 1988.

[19] R. E. Courtney and D. A. Gustafson, “Shotgun correlations in software measures,” *Software Engineering Journal*, vol. 8, no. 1, pp. 5–13, 1993.
[20] B. Kitchenham, S. L. Pfleeger, and N. Fenton, “Towards a framework for software measurement validation,” *IEEE Transactions on Software Engineering*, vol. 21, no. 12, pp. 929–944, 1995.
[21] N. Fenton and B. Kitchenham, “Validating software measures,” *Software Testing, Verification and Reliability*, vol. 1, no. 2, pp. 27–42, 1991.
[22] E. J. Weyuker, “Evaluating software complexity measures,” *IEEE transactions on Software Engineering*, vol. 14, no. 9, pp. 1357–1365, 1988.
[23] L. C. Briand, S. Morasca, and V. R. Basili, “Property-based software engineering measurement,” *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 68–86, 1996.