Assessing the Quality of Control Software Style Guides Used by Machine and Plant Manufacturers

Luis Steuter[1], Birgit Vogel-Heuser[1], *IEEE Fellow*, Dominik Hujo-Lauer[1] and Lucas Romier[1]

***Abstract*— In machine and plant manufacturing, control software is typically developed using the five programming languages defined in the IEC61131-3 and executed by Programmable Logic Controllers. Given current developments in automation, companies are faced with increasingly high software requirements, especially regarding software maintainability. A coding convention is essential to develop maintainable software. Although standardized style guides from PLCopen or PLC vendors exist, machine and plant manufacturers mostly create their own proprietary ones.**

Therefore, this paper compares 17 style guides from several companies and programming languages to identify differences and similarities and to retrieve a list of common style guide rules, with which industrial guides are assessed for their completeness, allowing software developers to categorically identify weaknesses. Style rule violations were quantified and tested in two industrial projects, generating meaningful software quality indications. Beyond, the paper offers explanations for the infrequent application of standardized guidelines. The findings facilitate the systematic improvement of PLC style guides and, thereby, software development processes in machine and plant manufacturing by exposing current weaknesses and deriving more mature principles from high-level programming.

***Index Terms*— automated production systems, software style guides, control software quality, software maintenance**

I. Introduction

Machine manufacturers are confronted with increasing software requirements, originating from high variability in their product line-up, a rising percentage of functionality realized in software, and lifecycles of multiple decades [1]. Developing maintainable software is therefore paramount. Scientific research and software developers agree on the importance of concise and consistent naming [2], high-quality commenting [3], and formatting elements [4] to increase readability and understandability and thereby foster software maintainability. The consistent implementation of these aspects is mostly enforced by coding guidelines, which are found to be essential for software maintainability [5]. Many companies, such as Google or Airbnb, developed their own guidelines and published and maintain them online. Subsequently, due to its widespread adoption among developers, e.g., Google's Java guideline, it emerged as one of the most popular coding conventions for writing Java code, cited by other companies and large open-source projects alike.

In machine and plant manufacturing, programming Programmable Logic Controllers (PLCs) with hard real-time capabilities according to one of the five languages defined in the IEC61131-3 is the current state of practice [6]. PLC vendors, such as market leader Siemens, develop and offer general guidelines for application software developers. Similarly, the organization PLCopen published a general, vendor-independent coding guideline specifically tailored for IEC61131-3 programming [7]. However, while 90% of companies in plant and machine manufacturing enforce guidelines, only 10% rely on PLCopen, with the remaining 80% utilizing their own standards [8], suggesting a high diversity of coding guidelines and practices inside industry.

Therefore, this paper introduces the calculation of completeness as a quality metric of a style guide in machine and plant manufacturing. By comparing different programming conventions, a set of individual style rules is compiled through which key differences and optimization potentials are identified and discussed.

The paper is organized as follows. Section II presents related work investigating naming, commenting, and formatting in software quality and the state of the art in PLC software quality assessment via static code analysis. Then, Section III compares guidelines from high-level and PLC programming to identify optimization potential and key aspects of PLC programming style guides. These aspects are then used in Section IV to introduce a quality metric for PLC control software style guides used by machine and plant manufacturers. Then, two industrial software projects are evaluated based on commonly enforced style rules.

II. State of the Art in Software Style Guides

While following a style guide is essential to develop high-quality software [5], not each common, individual rule regarding naming, commenting, and formatting is thoroughly studied [4]. An overview of the research on the impact of different guideline elements on software quality attributes is presented in Subsection II-A. Adherence to a style guide is commonly assessed when evaluating the quality of control software using static code analysis tools. Subsection II-B reviews their capabilities and relevance for the style guide analysis presented in this paper.

A. Naming, Commenting and Formatting for Code Quality

Individual style guide rules relating to naming, commenting, and formatting are mostly researched and proposed in textual programming for improving software maintainability by increasing readability and understandability, and thus source code quality.

Concise and consistent naming enables self-documenting code [2], which is paramount, since the source code itself

[1]The authors are with the Institute of Automation and Information Systems and head of the chair Prof. Dr.-Ing. Birgit Vogel-Heuser, TUM School of Engineering and Design, Technical University of Munich, Germany {luis.steuter, vogel-heuser, dominik.hujo, lucas.romier}@tum.de

is found to be the most important documentation artifact for maintenance-related tasks [9]. Stylistic aspects in naming, such as limiting the use of abbreviations or non-dictionary words and camel casing, have been shown to support software quality attributes [10]–[12].

Beyond identifiers, comments are found to be *very* important in understanding software for maintenance [9], although differentiating comment types and determining relevant factors and corresponding quality attributes continues to be an open research topic [3]. Several formatting elements, such as indentation [13], the number of statements per line, or blank spaces around operators and parameters [14] are studied from the perspective of software quality. However, research on formatting elements in code remains limited [4].

A selection of the investigated rules serves as necessary input for calculating metric values reflecting how readable and understandable a piece of code is, further underlining the crucial role of comments, identifiers, and formatting in software quality [15], [16].

However, most style guides are composed of more rules than have been currently researched or proven to benefit quality positively. This suggests that, while practices with a proven positive impact may be regarded as a necessary rule, a style guide, considered *complete* by software developers, needs to enforce additional elements.

B. Quality assessment in PLC software

Static code analysis tools can support the software development process in early stages. Prähofer et al. [17] applied static code analysis tools for IEC61131-3 and were able to check naming violations of a company-specific guideline, excessive complexity, or bad code smells. Fischer et al. [18] presented a set of conformance checks for IEC61131-3 software, which are configurable according to company-specific guidelines. Although naming violations are tested, the work focuses on more structural aspects, such as data exchange rules. Similarly, PLC vendors, such as Schneider Electric [19] or Beckhoff [20], offer functionalities for checking adherence to partly reconfigurable conventions. In Siemens' software development environment, the user can enable the automatic formatting of code, such as line indentations and conformance checks against Siemens' guidelines [21].

The introduced research and tools enable conformance checks to general and company-specific guidelines, including selected naming, commenting, and formatting rules. However, some implemented rules are similar or the same, while others are distinct, further emphasizing the variance in style guides. Beyond that, no existing approach assesses the company-specific conventions themselves for their completeness.

Therefore, the following work presents the comparison of guidelines from vendors, PLCopen, machine and plant manufacturers, and high-level programming to identify differences and common aspects to serve as a reference in determining the completeness of company-specific guidelines used in machine and plant manufacturing.

TABLE I
COMPARED STYLE GUIDELINES

Category for Table II	Abbreviation	Publisher/Industry	Language
High-level programming languages ***High-level***	GSC++	Google[1]	C++
	GSJ	Google[2]	Java
	GSPy	Google[3]	Python
	PEP8	Python[4]	Python
	AJS	Airbnb[5]	JavaScript
	ORA	Oracle[6]	Java
	CCS	Barr[7]	C
PLC vendors or organizations ***PLC v./o.***	PLCo	PLCopen [7]	IEC61131-3
	SIE	SIEMENS[8]	
	BECK	Beckhoff[9]	
	AB	Allen Bradley[10]	
	B&R	B&R Automation[11]	
Plant and machine manufacturers ***P.&M. M.***	ComA	Printing machines	IEC61131-3
	ComB	Intralogistic systems	
	ComC	Special purpose plants and machines	
	ComD	Packaging film production machines	
	ComE	Medical and pharmaceutical packaging machines	

[1] https://google.github.io/styleguide/cppguide.html
[2] https://google.github.io/styleguide/javaguide.html
[3] https://google.github.io/styleguide/pyguide.html
[4] https://peps.python.org/pep-0008/
[5] https://github.com/airbnb/javascript
[6] https://www.oracle.com/java/technologies/javase/codeconventions-contents.html
[7] https://barrgroup.com/sites/default/files/barr_c_coding_standard_2018.pdf
[8] https://support.industry.siemens.com/cs/document/109478084/programming-styleguide-for-s7-1200-s7-1500?dti=0&lc=en-DE
[9] https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_plc_intro/12049233675.html&id=
[10] https://literature.rockwellautomation.com/idc/groups/literature/documents/pm/1756-pm001_-en-e.pdf
[11] https://www.br-automation.com/en/academy/classroom-training/training-modules/programming-and-design-guidelines/

III. DIFFERENCES AND SIMILARITIES BETWEEN PLC AND HIGH-LEVEL PROGRAMMING

In this section, different software style guides from PLC and high-level programming are compared. First, a set of abstract style rules is gathered, with which differences and similarities between the considered guidelines are presented. Subsequent Subsections III-A to III-C offer explanations and details organized by category in naming, commenting, and formatting.

A total of 17 style guides were compared, which are provided in Table I. For each, the corresponding abbreviation, used in the scope of this paper and the publishing entity, is given. For plant and machine manufacturers ComA to ComE, the industry in which each company operates is provided, instead of the publisher. The rightmost column lists the corresponding programming language. Ten of the compared

TABLE II
SET OF ABSTRACT RULES RETRIEVED FROM GUIDELINE COMPARISON.
CELL COLORS REPRESENT PREVALENCE ON A CONTINUOUS SCALE RANGING FROM RED (0%) THROUGH YELLOW (50%) TO GREEN (100%).

Category	Subcategory	Number	Aspect	Details	Prevalence in		
					High-level	*PLC v./o.*	*P.&M. M.*
Naming	General	N1	Purpose	General purpose for naming	5/7	3/5	3/5
		N2	Language	Which language to use for identifiers	1/7	3/5	4/5
		N3	Length	Minimal or maximal length of a name	4/7	2/5	1/5
		N4	Uniqueness	Words/symbols to avoid	5/7	4/5	2/5
		N5	Abbreviations	Handling of abbreviations	3/7	4/5	2/5
		N6	Unique spelling	Some code elements can be identified solely from their spelling	7/7	3/5	5/5
	Content	N7	For data types	Suggests nouns or noun phrases	4/7	0/5	0/5
		N8	For functions and methods	Suggests verbs or verb phrases	4/7	2/5	0/5
		N9	Accordance to hardware (modules)	Naming has to correspond to hardware on module and device level	0/7	1/5	4/5
		N10	Commissioning signals	Rule for signals dedicated to commissioning	0/7	0/5	3/5
Commenting	General	C1	Formatting	Level of indentation and use of delimiters	6/7	4/5	2/5
		C2	Language	Regulates the language of comments	1/7	3/5	5/5
	Content and prerequisite	C3	Header comments	Content of header comments depending on the described element	5/7	3/5	2/5
		C4	Data types	How and when data types and their elements are commented.	6/7	3/5	3/5
		C5	POUs and methods	How and when functions, functions blocks and their arguments are commented.	6/7	4/5	3/5
		C6	Code blocks	How and when code blocks or FBD/LD-networks are commented.	4/7	4/5	4/5
		C7	Variables	How and when variables are generally commented	1/7	4/5	3/5
		C8	Variables corresponding to physical value	How and when variables with physical values are commented.	0/7	0/5	3/5
	Special comment types	C9	TODO-comments	Important elements of TODO-comments	2/7	0/5	2/5
		C10	Commented-out code	Treatment of commented-out code.	2/7	3/5	2/5
		C11	Nested comments	Treatment of nested comments	1/7	2/5	2/5
		C12	Inline comments	Placement, prerequisite and content of inline comments	7/7	4/5	2/5
Formatting	Indentation	F1	Amount	Amount of spaces to indent.	5/7	2/5	4/5
		F2	Tab character	Usage of “soft” or “hard” tabs.	6/7	3/5	3/5
	Line rules	F3	Maximum line length	Upper limit for characters per line	7/7	3/5	3/5
		F4	Statements per line	Upper limit for statements per line	5/7	1/5	1/5
		F5	Line break rules	How to break a break a line depending on the code	5/7	3/5	3/5
	Horizontal whitespaces	F6	For horizontal alignment	Regulates horizontal alignment for similar consecutive statements	4/7	1/5	1/5
		F7	Inside expressions or brackets	Regulates the use of whitespaces in/after statements, commas, elements.	6/7	4/5	0/5
	Special code elements	F8	Branching and looping statements	Where to put parentheses, spaces, line breaks, etc..	5/7	4/5	2/5
		F9	Type initializations	How to initialize types with many elements	5/7	3/5	0/5
		F10	Code blocks	Definition of code blocks and regulation of their separation	7/7	2/5	2/5
	Compatiblity	F11	Symbol coding	Allowed set of symbols in source files	3/7	1/5	1/5

style guides relate to PLC programming according to the IEC61131-3. Four of these are publicly available guidelines by PLC vendors, one originates from the standardization body PLCopen. The remaining five guidelines are internal guidelines, developed by machine and plant manufacturers. The companies cover industries from automated intralogistics to medical and pharmaceutical packaging machinery, offering broad sector coverage (cf. Table I).

A positive correlation between the control software quality and object-oriented programming, version control via Distributed Version Control Systems, as well as the usage of the textual language Structured Text, has been identified [22]. Therefore, more companies are expected to adapt their development process and software accordingly. Because these concepts are currently not widely or only recently applied in machine and plant manufacturing, seven commonly

used high-level programming style guides are considered for comparison (cf. Table I), anticipating more mature and detailed conventions, since these principles have been firmly established in languages like Java or C++.

The aspects listed in Table II represent a set of abstract style rules. They are abstract. since their concrete implementation can differ. For instance, the formatting rule F3 concerns an upper limit of characters per line. The concrete realization of this rule differs across the compared style guides, with e.g. Google's Python guide (GSPy) prescribing an upper limit of 80 characters, ComA or B&R Automation (B&R) of 120 characters.

The prevalence of a given rule in each style guide category from Table I is listed in the three rightmost columns in Table II. For its determination, the concrete realization is not considered. For example, for formatting rule F2 regarding the tab character, ComC requires the use of tabs with the actual tab character to indent code (i.e., "hard tabs"), while ComA and ComD specify the number of spaces and prohibit the use of the tab character (i.e., "soft tabs"). Because ComB and ComE do not mention with which characters to indent code, the prevalence among plant and machine manufacturers (***P.&M. M.***) amounts to three from five, as summarized in the corresponding entry in Table II and colored accordingly.

A. Naming

In most guidelines across all categories, a general purpose for naming is proposed (N1) along with spelling rules to differentiate code elements solely from their spelling (N6), such as reserving UPPER_SNAKE_CASE-spelling uniquely for constants. However, the considered PLC vendors and PLCopen suggest a large number of uniquely spelled elements, e.g., more than 20 unique elements in PLCopen. Companies from PLC and high-level programming only adopt a few of those, at most *nine* in the case of Company E. Commonly, the minimal length of an identifier is discussed (N3) or an explicit character range is stated, such as three to 31 characters per identifier in *Barr's C Coding Standard* CSS.

Rules N9 and N10 can be uniquely found in guidelines from machine and plant manufacturers. Most insist on a clear relation between the software identifiers and the mechanical machine layout or the electrical wiring diagram (N9). Special prefixes are imposed on signals used during machine or plant commissioning to distinguish and, if applicable, automatically identify and eliminate these signals once commissioning is complete (N10).

High-level programming and PLC programming guidelines diverge on the concrete implementation of which words and symbols to avoid (N4). Most PLC programming guidelines specifically address the issues of "shadowing" a local with a global variable name and including the function block type in a function block instance. Whereas high-level programming guidelines specify the symbol set, a name's characters need to originate from, such as ASCII (cf. PEP8 or GSJ), and give concrete lists with words to avoid.

Commonly, high-level programming guidelines suggest using nouns or noun phrases for data types, such as classes and verbs or verb phrases for functions or methods (N7 & N8). This rule is not implemented by any of the considered PLC guidelines. Potential benefits are yet to be explored, but can be anticipated in PLC programming, especially when object orientation is used.

B. Commenting

In commenting, while rules are similarly prevalent in PLC programming as in high-level programming, the concrete implementation shows significant differences. The compared PLC programming guidelines are generally not differentiating between commenting for different types of elements, stating rather generic rules, such as "all elements shall be commented" (PLCo) [7], therefore implicitly mentioning rules C4 to C7 and C12. High-level programming guidelines provide more details on which elements shall be commented in what manner.

File header comments (C3) serve as an important source of information in most high-level programming with e.g., license boilerplates, summaries, or usage examples. In contrast, only two of five machine and plant manufacturers enforce a similar file header comment structure, suggesting potential for improving reusability and code understandability.

Special comment types (C9 to 11) are mentioned similarly frequently across all guidelines. Most permit nested comments and specify the structure and key elements when commented-out code is left, or TODO comments are written. Three of five machine and plant manufacturers exclusively consider how to comment variables corresponding to real-world values, prescribing, e.g., that the comment comprises the unit.

C. Formatting

Most guidelines agree on using "soft tabs", i.e., spaces, instead of "hard tabs", i.e., the tab character for code indentation (F1), while the indentation amount (F2) ranges between two and four characters per level.

Significant differences can be observed for rules F4 to F10. Generally, PLC programming guidelines mention these programming rules less frequently, and, if mentioned, in far less detail. High-level guidelines strictly forbid the horizontal alignment of similar, consecutive code lines (F6) to avoid extra work in development and code review, while ComD promotes it, which, judging by the prevalence in high-level programming, is most likely the inferior approach.

A maximal line length (F3) is enforced by most guidelines, while the actual number of characters ranges between 72 and 130. Since the IEC61131-3 defines graphical languages as well, Beckhoff (BECK), for example, sets the maximal size of a graphical programming block, i.e., a network, to be fully visible without scrolling horizontally. Generally, the horizontal extension of graphical source code is relatively difficult to enforce, and a solution independent from the developer's screen size is not implemented in any of the considered guidelines.

IV. Style Guide and Software Style Analysis

The following section introduces the metric of completeness for a given style guide in machine and plant manufacturing, which is calculated relative to the retrieved rules from Table II. The completeness is computed for each industrial coding convention from ComA to ComE and compared. Then, in Subsection IV-B, two projects from ComA are analyzed regarding their adherence to the presented rules.

A. Style Guide Assessment

By considering guides from different programming languages, PLC vendors, and software developers from a broad range of industries, the retrieved aspects in Table II represent a *unionized* style guide, which naturally comprises more elements than the compared coding conventions from Table I. Thereby, a metric for completeness can be computed with the number of rules the company has implemented, which are part of Table II. For instance, ComA has implemented

- in Naming: N1 through N6 and N9,
- in Commenting: C1 through C5, C7, C8 and C11
- and in Formatting: F3, F10 and F11.

This amounts to 18 implemented rules by ComA. Divided by the number of total rules in Table II a completeness of

$$\frac{18}{33} = 54.55\%.$$

results. Currently, each rule equally influences the completeness. With a weighing mechanism (i.e., from the prevalence in Table II), a more accurate quality metric could be derived.

Analogously, a completeness can be determined for each subcategory. For instance, ComC regulates the language for identifier names (N2), which words to avoid (N4), states capitalization rules (N6) and specific prefixes to identify commissioning signals (N10). Thus, four of the ten, or 40% of the naming rules (cf. Table III) are implemented.

The results for all five companies are presented in Table III. The most complete style guide is the one from ComC, which is especially detailed in how and why to comment different code elements. In aspects related to Naming ComA displays the best metric value. In Formatting, not a single style guide scores above 60%, reinforcing previous observations from Subsection III-C.

TABLE III
Style Guide Completeness for Each Company

Company	Completeness			
	Total	Naming	Commenting	Formatting
ComA	**54.55%**	70%	66.67%	27.27%
ComB	**18.18%**	30%	25%	0%
ComC	**63.63%**	40%	91.67%	54.55%
ComD	**33.33%**	30%	25%	45.45%
ComE	**60.61%**	60%	83.33%	36.36%

B. Software Project Style Adherence

The rules N1 through N8 are applied in two software projects, Project 1 and 2 from ComA. To each rule, a percentage value is assigned per project, resulting in Figure 1. The calculation method for each value is given in Table IV. Most identifiers are made up of more than one word. Some rules are calculated relative to the total number of words, some to the number of identifiers. Additionally, Table IV describes which method is based on the concrete implementation from the company-specific guideline. If a rule is not implemented, a common practice from other guidelines is applied. Automatic assessment is feasible for most of the rules from Table II.

TABLE IV
Computation of Each Value for Figure 1

Rule	Value calculation	In ComA
N1	Percentage of identifiers with more than 4 characters with at least one word in the english dictionary	yes
N2	Percentage of words in the english dictionary	yes
N3	Percentage of identifiers with at least 4 characters	yes
N4	Percentage of identifiers not using standard function names or keywords	yes
N5	Percentage of identifiers with no abbreviations	yes
N6	Percentage of identifiers adhering to UpperCamelCase (or UPPER_SNAKE_CASE for constants)	yes
N7	Percentage of identifiers with first word a noun	no
N8	Percentage of identifiers with first word a verb	no

The two projects implement different modules from the same machine. In total, Project 1 consists of 32,484 identifiers totaling 86,803 words, and Project 2 of 19,886 identifiers with 49,531 words. Software developers from ComA assign lower quality to Project 1 compared to Project 2.

Project 2 scores better for each value in Figure 1, except for N4, agreeing with the quality assignment from the company experts. For rule N7 and N8, Project 2 exhibits a higher value as well, although ComA does not specifically recommend these, hinting at a correlation with software quality. The analysis of comments and formatting elements in both projects shows similar results.

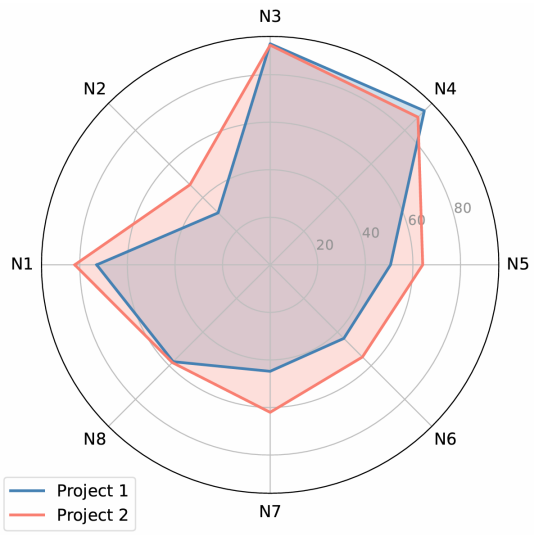

Fig. 1. Adherence to rules N1 through N8 for two projects from ComA

C. Discussion

The comparison of guidelines showed that differences between PLC and high-level programming exist, which makes the direct application of high-level style guides in machine and plant manufacturing impossible. However, with certain elements not defined in PLC programming guidelines, such as recommending nouns for naming data types, meaningful quality indicators could be generated for comparing two industrial software projects.

While high-level programming guidelines were found to be generally more detailed, the most significant disparity in detail depth was observed in formatting elements. One possible explanation for this is the origin of PLC programming from graphical languages. Since PLC software developers historically had less freedom in formatting their (graphical) pieces of software, the importance of formatting in itself does not seem to be generally recognized yet.

Standard coding conventions, created by vendors or PLCopen, are not commonly enforced in machine and plant manufacturing. They are referenced, but always tailored to a specific company's needs, with selected prefixes, commissioning signals, or header comments. While some company guidelines (`ComC` and `ComE`) were found to be more detailed than PLCopen, others, such as `ComB`, exhibited a considerable deficit, underlining the variance across industry.

V. Conclusion and Outlook

This paper introduced a metric for computing the completeness and categorically identifying weaknesses of a style guide in machine and plant manufacturing. The metric computation is based on the retrieval of common style guide rules, gathered from 17 programming guidelines. Rule violations were automatically assessed in two industrial software projects, aligning with the experts' quality judgments. Systematic improvement of PLC software guidelines and development processes is thereby facilitated by exposing weaknesses and proposing more mature rules from high-level programming.

Building on this contribution, more quality attributes for coding conventions used by machine and plant manufacturers can be developed. For instance, each element can be weighted according to its importance to provide an improved quality value. Beyond that, implementation details can be taken into account to measure correctness. Some aspects can be sufficiently argued for due to their prevalence and practical relevance. Others lack scientific evidence or a formal basis in machine and plant manufacturing and need to be investigated more thoroughly.

References

[1] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, "Evolution of software in automated production systems: Challenges and research directions." *J. Syst. Softw.*, vol. 110, no. 110, pp. 54–84, 2015.

[2] F. Deissenboeck and M. Pizka, "Concise and consistent naming," *Software Quality Journal*, vol. 14, no. 3, pp. 261–282, 2006.

[3] P. Rani, A. Blasi, N. Stulova, S. Panichella, A. Gorla, and O. Nierstrasz, "A decade of code comment quality assessment: A systematic literature review," *Journal of Systems and Software*, vol. 195, p. 111515, 2023.

[4] D. Oliveira, R. Santos, F. Madeiral, H. Masuhara, and F. Castor, "A systematic literature review on the impact of formatting elements on code legibility," *Journal of Systems and Software*, vol. 203, p. 111728, 2023.

[5] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "Learning natural coding conventions," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 281–293.

[6] B. Vogel-Heuser, J. Fischer, S. Feldmann, S. Ulewicz, and S. Rösch, "Modularity and architecture of PLC-based software for automated production systems: An analysis in industrial companies," *Journal of Systems and Software*, vol. 131, pp. 35–62, 2017.

[7] PLCopen, "PLCopen Coding Guidelines, Version 1.0," PLCopen, Tech. Rep., Apr. 2016, official release (Apr. 20, 2016), 127 pages. [Online]. Available: https://www.plcopen.org/download_file/force/ff60e817-eee5-442d-b718-a357a7279c7e/342/

[8] B. Vogel-Heuser and F. Ocker, "Maintainability and evolvability of control software in machine and plant manufacturing—an industrial survey," *Control engineering practice*, vol. 80, pp. 157–173, 2018.

[9] S. C. B. De Souza, N. Anquetil, and K. M. De Oliveira, "A study of the documentation essential to software maintenance," in *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*, 2005, pp. 68–75.

[10] D. Binkley, M. Davis, D. Lawrie, J. I. Maletic, C. Morrell, and B. Sharif, "The impact of identifier style on effort and comprehension," *Empirical Software Engineering*, vol. 18, no. 2, pp. 219–276, 2013.

[11] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Exploring the influence of identifier names on code quality: An empirical study," in *2010 14th European Conference on Software Maintenance and Reengineering*. IEEE, 2010, pp. 156–165.

[12] J. C. Hofmeister, J. Siegmund, and D. V. Holt, "Shorter identifier names take longer to comprehend," *Empirical Software Engineering*, vol. 24, no. 1, pp. 417–443, 2019.

[13] J. Bauer, J. Siegmund, N. Peitek, J. C. Hofmeister, and S. Apel, "Indentation: simply a matter of style or support for program comprehension?" in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 154–164.

[14] I. B. Sampaio and L. Barbosa, "Software readability practices and the importance of their teaching," in *2016 7th International Conference on Information and Communication Systems (ICICS)*. IEEE, 2016, pp. 304–309.

[15] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto, "Automatically assessing code understandability," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 595–613, 2019.

[16] R. P. Buse and W. R. Weimer, "Learning a metric for code readability," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 546–558, 2009.

[17] H. Prähofer, F. Angerer, R. Ramler, and F. Grillenberger, "Static code analysis of IEC 61131-3 programs: Comprehensive tool support and experiences from large-scale industrial application," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 1, pp. 37–47, 2016.

[18] J. Fischer, B. Vogel-Heuser, F. Haben, L. Beuggert, and E.-M. Neumann, "Towards configurable conformance checks of PLC software with company-specific guidelines," in *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2022, pp. 01–08.

[19] *EcoStruxure Machine Expert - Code Analysis, User Guide*, Schneider Electric, Dec. 2025, accessed on: 2026-02-04. [Online]. Available: https://www.se.com/us/en/download/document/EIO0000002710

[20] *TwinCAT 3 - PLC Static Analysis*, Beckhoff Automation GmbH & Co. KG, Jan. 2026, accessed on: 2026-02-04. [Online]. Available: https://download.beckhoff.com/download/document/automation/twincat3/TE1200_TC3_PLC_Static_Analysis_EN.pdf

[21] *Project Check for TiA Portal*, Siemens, Jan. 2023, accessed on: 2026-02-04. [Online]. Available: https://support.industry.siemens.com/cs/attachments/109741418/109741418_ProjectCheck_Intro_DOC_V3.1.0_en.pdf

[22] B. Vogel-Heuser, E.-M. Neumann, and J. Fischer, "Maturity levels for automation software engineering in automated production systems," in *2022 IEEE 20th International Conference on Industrial Informatics (INDIN)*. IEEE, 2022, pp. 618–623.