Applications

Quang Huan Dong*, Birgit Vogel-Heuser and Eva-Maria Neumann

Semi-automatic assessment of lack of control code documentation in automated production systems

Semiautomatische Bewertung fehlender Dokumentation von Steuerungscode in automatisierten Produktionssystemen

A risk-based approach to indicate documentation debt

Ein risiko-basierter Ansatz zur Indikation von Dokumentationsschulden

https://doi.org/10.1515/auto-2022-0146
Received November 11, 2022; accepted May 4, 2023

Abstract: This paper first examines the current state of industrial practice of documentation in automated production systems based on a large-scale survey in machine and plant manufacturing proving that companies still face major challenges in documentation. However, insufficient documentation creates friction since it may increase the risk of malfunction and high costs, and impede system development due to lack of traceability, especially for control software being one of the main functionality carriers. Therefore, secondly, a risk priority indicator RPI4DD is proposed to systematically capture the lack of control code documentation to avoid undesired costs due to inadequate documentation.

Keywords: automated production systems; documentation; machine and plant manufacturing; risk analysis; risk priority indicator; software engineering.

1 Introduction and motivation

In factory automation, machine (MM) and plant manufacturing (PM) companies are facing numerous challenges, such as high variability, Industry 4.0 requirements (e.g., small lot sizes) or long lifecycles up to decades [1, 2]. German manufacturers in this area, who used to be world-leading exporters, are confronted with growing competition worldwide and thus are struggling to stay competitive, particularly regarding work costs. Some MM and PM companies have to consider relocating engineering divisions to lower-wage countries with less experienced/qualified engineers who need more mature documentation [3]. Moreover, the scope of system functionalities realised by software is increasing, leading to a growing software complexity [4].
Thus, companies need to improve complexity management methods for software.

Fischer et al. [5] indicate that software complexity strongly influences comprehensibility (readability), a prerequisite to maintaining or reusing existing code. Thus, necessary documentation artefacts (e.g., architecture documents or code comments) are required to ease the readability of complex software. The impact of insufficient documentation might be significant, especially among low-skilled engineers in lower-wage countries where the engineering departments are relocated [3]. A motivational example is introduced in the following:

In the commissioning phase […] an error handling routine for a pneumatic cylinder is missing […] In the optimal case, a change request to the programmer, who developed the library function for controlling pneumatic cylinders, should be made. However, as the time pressure to start the [systems] is high, the commissioner implements the error handling on the next higher software architecture level that he may access, i.e., directly in the application, since the library function is not accessible for him […] This conscious decision of avoiding proper change management and violating the architectural concept is often accompanied by a lack of documentation. As the commissioner works as fast as possible to start up the [systems], [she/he] does not document the changes made in the software resulting in many software versions in the field [6].

Motivated by the example above, a question arises: How do different factors influence control code documentation? Since the development characteristics of MM and PM are typically distinct, they often follow different engineering practices. Previous studies indicated that software analysis alone is insufficient to measure software quality (e.g., maturity level [7]) or support software evolution since automation software is part of a mechatronics system involving multiple disciplines or stakeholders [8]. Instead, the development process characteristics must be considered. Therefore, a web questionnaire is developed to study software maturity, complexity, and documentation in MM and PM [7].

The main contribution of this work is to address selected aspects of (semi-) automatically identifying a lack of control code documentation. First, based on the results of [7], a study is conducted to analyze the documentation aspect from expert responses to the questionnaire, especially regarding engineering practices or the automatic generation of information in MM and PM. Second, a concept to assess risk associated with insufficient documentation is proposed, and the concept’s applicability to control code documentation is presented for supporting an early reaction or counter-measures in the development workflow.

The remainder of the paper is structured as follows. Section 2 provides the background and related work, followed by an analysis of industrial practice regarding documentation in MM and PM in Section 3. The concept and applicability of a risk-based approach to indicate insufficient documentation are presented in Section 4. Finally, Section 5 concludes the paper and provides an outlook on future work.

2 Background and related work

This section introduces some backgrounds of the systems developed by MM and PM so-called automated production systems (aPS), according to [8], followed by a description of the quality analysis of aPS software. The section continues with a discussion on the control code documentation. Concluding, the research gap is presented.

2.1 Development of aPS

aPS is developed by engineers from multiple disciplines [8]. The development often starts in the mechanical engineering discipline, which creates the construction plan of the mechanical parts. Based on this construction plan and component lists (e.g., sensors or actuators), the electrical engineers design the electrical system. Documents from both disciplines are then forwarded to the software engineers, who use them to develop the software running in Programmable Logic Controllers (PLC) (i.e. control software or aPS software), the hardware platforms used to manage the automation for machines and plants via sensor inputs and actuator outputs.

There are differences in the languages for PLCs compared to those used in classical software engineering. The aPS software is often developed following the IEC 61131-3 standard [5]. IEC 61131-3 defines three types of Program Organization Units (POU): (1) Function (FC), (2) Function Block (FB), and (3) Program (PRG). Each POU includes a comment header, a variable declaration section, and an implementation section. IEC 61131-3 compliant languages include three graphical languages (i.e., Structured Text ST and Instruction List IL). In the field of PLC programming, there are guidelines from the communities of PLCopen or MISRA [9]. PLCopen is a worldwide initiative of different platform suppliers to reduce engineering effort and enhance software quality by providing standards, guidelines, and education for platform users in the community of industrial automation. The PLCopen guidelines, e.g., naming conventions or structuring with SFC, support in ensuring, e.g., consistency of automation software. MISRA provides rules
and directives for coding and documentation that improve software maintainability. The MISRA guidelines, e.g., using start and end comment markers, support in, e.g., portability of automation software.

The aPS development is often validated with GAMP (Good Automated Manufacturing Practice) [10], which is a risk-based approach widely used in industry to regulate computerised systems. Risk Priority in the GAMP method (i.e., criticality estimation in risk analysis) is based on three factors: Severity, Probability, and Detectability (cf. Figure 1). For each risk entry, ratings are given to each of the factors. Risk Priority is calculated with a multiplicative approach: factors are multiplied by each other in two stages. The factors are allocated to two levels: Detectability and Risk class are level 1 factors; Severity and Probability are level 2 factors. In the first stage, the Severity is offset against the Probability. This calculation determines the Risk class (1–3). In the second stage, the Risk class is then offset against the Detectability and results in a Risk Priority (low, medium, high).

### 2.2 Quality analysis of aPS software

In classical software engineering, various approaches are available to measure software characteristics, such as complexity or code documentation (i.e., code comments). It is due to high complexity or poor code documentation that might hinder software maintainability, which may introduce undesired additional costs [11]. Analogously, characteristics of aPS software need to be analysed to measure the criticality of documentation for the software. Vogel-Heuser et al. [12] proposed the MICOSE4aPS approach to measure the maturity of software considering different change types (e.g., bug fixing or new development). Wilch et al. [13] presented a semi-automatic concept to identify POU functionality based on characteristics of implementation or description. Fischer et al. [5] proposed an approach to compare the complexity of POUs using a metric for the Overall Complexity based on six sub-metrics, $M_1$–$M_6$, as follows.

- ProgramLength ($M_1$) refers to Halstead’s Program Length, i.e., the sum of the tokens (operators and operands).
- Cyclomatic Complexity ($M_2$) refers to McCabe’s Cyclomatic Complexity determined by the number of decision statements.
- FanIn_FanOut ($M_3$) is determined by incoming (e.g., input variables) and outgoing (e.g., output variables) data flows.
- Vocabulary Size ($M_4$) and Difficulty ($M_5$) follow the ideas of Halstead’s Vocabulary Size and Difficulty, which are also based on the number of (unique) operators and operands.
- Data Structure ($M_6$) targets the complexity of the data processed. For instance, the program interface variables add more complexity than local variables and sub-variables.

The calculation includes three steps. First, six metrics $M_1$–$M_6$ ($M_i$ in (1)) are applied to each POU, resulting in six values with different scales per unit. The median values $\tilde{M}_1$ to $\tilde{M}_6$ are then determined. Second, the metrics values are scaled using equation (1) to the corresponding median.

$$C_{rel,M_i}(POU) = \frac{M_i(POU)}{\tilde{M}_i} \cdot 100\%$$

Third, the Overall Complexity $OC_{rel}$ of a POU is calculated based on the sum of the six metrics values scaled in the second step (cf. equation (2)). The weights $w_i$ are used to adjust the influences of individual metrics on the overall complexity. The sum of $w_i$ is 1 to scale Overall Complexity values to a predefined value range (e.g., (0, 1)).

$$OC_{rel}(POU) = \sum_{i=1}^{6} w_i \cdot C_{rel,M_i}(POU)$$

### 2.3 Lack of control code documentation in the aPS domain

In classical software engineering, the phenomenon of insufficient documentation is referred to as technical debt (TD), more specifically, documentation debt [14]. The TD concept describes a context where a technical compromise is selected against better knowledge [15]. For instance, low-quality code is delivered to meet an urgent deadline. The technical compromise can enable a short-term benefit (e.g., quicker delivery) but may introduce some undesired effects (e.g., high maintenance cost). The survey of Li et al. [15] indicates that although documentation debt received significant attention, studies on documentation debt management are scarce. Avergiou et al. [16] reported that TD always relates to cost. Inadequate monitoring of TD may introduce additional long-term effort in software projects. TD might cause project delivery delays, as well as a decrease in organisational profitability. Detofeno et al. [17] suggested that to prioritise TD in software, one should not only consider source code alone but also consider external factors such as project characteristics or test coverage.
Near the aPS domain, Martini et al. [18] analysed the accumulation of TD in software in embedded systems and additional activities (e.g., refactoring) required at five software companies. They reported that insufficient documentation “causes the misinterpretation by the developers implementing code” as well as hinders refactoring activities. However, code comments were not considered as the work focused on the level of design and architecture. A survey with six software companies from Besker et al. [11] indicated that time pressure commonly triggers documentation debt, such as insufficient updating related documentation for code implementation. The study reported a significant productivity loss of software developers due to TD. However, the code comment was not focused.

In the aPS domain, Besker et al. [14] investigated the work of software engineers at one company working with aPS in Scandinavia and reported that there were “quite a lot of resources in paying the interest on Technical Debt, on average 32% of the development time”. Biffl et al. [19] studied the risks of TD in engineering artefacts related to data exchange (e.g., process documentation and configuration management). Waltersdorfer et al. [20] reported two types of documentation debt: Insufficient Data Model and Insufficient Product Process Resources Model. However, the code comment was not considered in the above studies.

Due to the nature of aPS, on-site changes are often performed to adapt the manufacturing systems to unanticipated raw materials or environmental conditions. Under pressure to start up the system, proper control code documentation for on-site adaptations might be neglected, resulting in documentation debt (cf. Section 1, e.g.). TD is causing additional development and maintenance costs over time [14] or developer productivity loss [11]. Due to the long lifetime of aPS, the impact from TD might be large. Unfortunately, awareness of TD in the aPS domain is still low in general, according to a survey of 48 German companies supplying aPS [21].

2.4 Research gap

Research addressing aPS documentation quality is still scarce. Among the publications, Neumann et al. [22] introduced templates to document design decisions at the architectural level of aPS. Neumann et al. [23] proposed some simple metrics such as HeaderCommentLinesOfCode (size of comment header), MultiLineComments (amount of comments wrapping over multiple lines), and SourceCodeCommentedRatio (density of comments) to assess the software maintainability. The authors’ previous work [7] mainly focused on methods for document exchange or on how code/configuration is generated from documents.

As aforementioned, to assist the engineers in maintaining, enhancing, or reusing the control code at a later stage, the comment header often provides an overview description of the POU, and the comments in the variable declaration and implementation section explain the control code in detail. A well-documented software supports its reusability in future projects; thus, the development costs could be optimised. Not only code comments, but a broad view should also be considered due to interdisciplinary constraints. Thus, the availability of documentation in different disciplines, the use of internal guidelines, or the automatic generation of engineering documents needs to be studied. Another factor in enabling reusability is standardisation, i.e., whether standardised guidelines provided by the community are followed. To the best of the authors’ knowledge, there is no method to determine the criticality or quality of documentation in the aPS domain. By analysing results from an industrial expert survey in machine and plant manufacturing, this work firstly studies the state of the practice regarding documentation. Secondly, factors influencing the documentation activities are derived considering the documentation debt concept to propose a risk priority indicator for identifying the documentation debt. An overview of findings from related work and expert survey corresponding to the sections is illustrated in Figure 2.

3 An analysis of state of the practice regarding documentation

The current state of practice in industry regarding documentation was surveyed using a questionnaire distributed via newsletters and web pages addressing a German-speaking community interested in embedded systems, mechanical, electrical or software engineering in MM and PM. For this paper, the focus is on the documentation aspect of the questionnaire, which was not the target of the previous investigation [7]. Hereafter, #number denotes a question with the ordinate number in the questionnaire available online [24]. The eight questions used in this analysis are listed in the Appendix. Among 322 companies that started to answer the questions, 146 finished it. As this work targets aPS, those 146 completed cases are firstly categorised by their answers regarding industrial sectors (#65). Thus, only responses from companies assigning themselves to at least one of the sectors associated with MM and PM, such as material handling or woodworking machinery, are selected. This reduces the cases from 146 to 71. Second, only companies assigned themselves to MM or PM in #69 are considered,
thus reducing the cases from 71 to 61 companies, representing the study’s base data. In the following, the results of the documentation aspect are reported.

As a pre-processing step, question #66 surveyed the company size. Question #72 was used to study the functional description availability in disciplines. The availability of documents is lowest in software compared to electrical/electronics and mechanical disciplines (cf. Table 1). Across the disciplines, the availability is highest at large companies (>1000 employees) and lowest at small-and-medium companies (from 50 to 250 employees). The availability of documents “when design begins” or “provided at milestones” is lowest in software (cf. Table 1). Thus, document availability is mostly poor when engineering begins and varies later depending on the discipline and company size.

Questions #51 and #52 are used to study how engineering artefacts are automatically generated in different disciplines (e.g., code generation or configuration). The identified sources reveal that requirements documents are most commonly used across all disciplines (cf. Table 2); however, the usage frequencies are still low. Thus, automatically generating engineering artefacts is still poor in general.

Following guidelines or checklists in software engineering supports high-quality code and traceable software [9]. Question #77 studied the differences regarding the levels of detail of internal guidelines in different disciplines. Software and electrics/electronics received more precise instructions (21 % and 20 %, respectively) than mechanics (11 %) (cf. Table 3). However, the formulation of guidelines is still poor in general.

Question #34 examined the usage of PLCopen and company-specific guidelines. The responses of #34 show a low frequency of usage of PLCopen guidelines (10 %) compared to in-house guidelines (cf. Table 4). The internal

<table>
<thead>
<tr>
<th>Availability of documents</th>
<th>Software</th>
<th>Electrics/electronics</th>
<th>Mechanics</th>
</tr>
</thead>
<tbody>
<tr>
<td>When design begins</td>
<td>9 %</td>
<td>13 %</td>
<td>13 %</td>
</tr>
<tr>
<td>Often late</td>
<td>28 %</td>
<td>29 %</td>
<td>25 %</td>
</tr>
<tr>
<td>Provided at milestones</td>
<td>25 %</td>
<td>31 %</td>
<td>25 %</td>
</tr>
<tr>
<td>Engineer needs to request</td>
<td>20 %</td>
<td>8 %</td>
<td>13 %</td>
</tr>
<tr>
<td>Engineer has to collect details from customer</td>
<td>3 %</td>
<td>4 %</td>
<td>3 %</td>
</tr>
<tr>
<td>Not applicable</td>
<td>15 %</td>
<td>15 %</td>
<td>20 %</td>
</tr>
</tbody>
</table>
Table 2: Evaluation of expert surveys on automatically generating engineering artefacts across disciplines (#51 and #52 [24]).

<table>
<thead>
<tr>
<th>Source for automatic generation</th>
<th>Software</th>
<th>Electrics/electronics</th>
<th>Mechanics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements documents</td>
<td>38 %</td>
<td>29 %</td>
<td>26 %</td>
</tr>
<tr>
<td>Component lists</td>
<td>–</td>
<td>11 %</td>
<td>11 %</td>
</tr>
<tr>
<td>E-Plan (a)</td>
<td>–</td>
<td>17 %</td>
<td>7 %</td>
</tr>
<tr>
<td>Models</td>
<td>12 %</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Total from above artefacts</strong></td>
<td><strong>50 %</strong></td>
<td><strong>57 %</strong></td>
<td><strong>44 %</strong></td>
</tr>
<tr>
<td>Nothing</td>
<td>16 %</td>
<td>17 %</td>
<td>18 %</td>
</tr>
</tbody>
</table>

\(a\)E-Plan: a tool for electrical engineering of machines and plant systems.

Table 3: Evaluation of expert surveys on detail levels of in-house guidelines across disciplines (#77 [24]).

<table>
<thead>
<tr>
<th>Detail level of internal guidelines</th>
<th>Software</th>
<th>Electrics/electronics</th>
<th>Mechanics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor in-house guidelines</td>
<td>33 %</td>
<td>36 %</td>
<td>44 %</td>
</tr>
<tr>
<td>Fine-grained in-house guidelines</td>
<td>31 %</td>
<td>29 %</td>
<td>24 %</td>
</tr>
<tr>
<td>Precise instructions</td>
<td>21 %</td>
<td>20 %</td>
<td>11 %</td>
</tr>
<tr>
<td>Not applicable</td>
<td>15 %</td>
<td>15 %</td>
<td>21 %</td>
</tr>
</tbody>
</table>

Table 4: Evaluation of expert surveys on usage of PLCopen and company-specific guidelines in software discipline (#34 [24]).

<table>
<thead>
<tr>
<th>Types of guidelines in use</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLCopen guidelines</td>
<td>10 %</td>
</tr>
<tr>
<td>In-house guidelines</td>
<td>80 %</td>
</tr>
<tr>
<td>None</td>
<td>10 %</td>
</tr>
</tbody>
</table>

Guidelines are employed by a large portion of the surveyed companies (80 %); thus, the usage of guidelines from the community is less than company-specific guidelines. Furthermore, the 10 % remaining responses reported that no guideline is being followed.

In summary, the results show a lack of availability of required documents, low-moderate automatic generation of information in engineering, a lack of exact instructions, and high reliance on in-house guidelines in MM and PM. These findings reveal the potential triggers of documentation debt in aPS. Thus, a methodology to indicate the risk of documentation debt is necessary.

4 Concept of indicators for control code documentation debt

The expert responses presented in Section 3 emphasise that insufficient documentation is still a critical challenge in industrial practice of MM and PM companies. The quantitative results in Section 3 are substantiated by a qualitative study [25] conducted by the authors with an industrial partner developing aPS for the healthcare industry. The results indicate a high need to improve document availability as an essential to build up a cross-disciplinary development process. Especially in software, which must be adaptable over many years and is increasingly becoming one of the main carriers of system functionality, a lack of documentation is a significant cost factor. However, to systematically improve the software documentation, it is first necessary to quantify the risk of insufficient documentation and thus to assess where an improvement in documentation is most urgent to prevent errors at an early stage, reduce engineering duration, and shorten the time-to-market. Therefore, a Risk Prioritisation Indicator of Documentation Debt (i.e. RPI4DD) is introduced in the following to quantify the risk of a lack of control code documentation.

This section first derives requirements of indicators for documentation debt, then presents an approach to transfer the concept of RPI4DD, followed by a set of selected factors to indicate control code documentation debt in aPS. The selected factors are based on current literature and expert responses identified in Section 3. The section continues with a description of the proof of concept of RPI4DD, including an evaluation using a lab-sized plant. Finally, a summary of RPI4DD calculation and requirement fulfilment is given.
4.1 Derived requirements of indicators for documentation debt

From the state of the art, requirements of indicators for documentation debt are derived in the following.

**R1. RPI4DD should be extensible with new factors identified.** The requirement for extensibility is due to TD research in the aPS domain being an ongoing work. The aspects of identifying TD, in particular documentation debt, are not fully explored yet. Thus, the list of factors is not yet fixed. When new factors are identified in future work, their calculation rules should be seamlessly integrated into RPI4DD.

**R2. RPI4DD should be flexible with calculation rules of individual factors.** Different calculation rules might be introduced to assess a factor due to the high diversity of applications in the context of the aPS domain. Therefore, flexibility is a requirement for RPI4DD.

**R3. RPI4DD should be automatable.** The program size of industrial projects is often large; thus, an automatic method is required to enable applicability in industrial practice.

4.2 Concept of a risk-based approach to indicate documentation debt

The concept of determining indicators for documentation debt is based on the GAMP method, which allows extensibility for additional factors that could be added to an appropriate level. Therefore, since the GAMP approach satisfies **R1 (extensible)**, the approach is used as a basis to derive RPI4DD. To satisfy **R2 (flexible)**, weights could be added to reflect the different importance of each factor. In addition, besides the multiplication operator, other operators (e.g., addition) could also be applied to the calculation of the factors.

In the following, based on the GAMP method, the six main steps to establish a new risk assessment method for documentation debt (cf. Figure 3) are described. This work focuses as a starting point on the control code documentation, in particular code comments and naming conventions. However, the design of RPI4DD shall enable the integration of further documentation artefacts, such as manuals or architect documents, in future work (cf. R1).

- **Step 1: Identify an initial set of factors influencing TD.**
  In documentation in general and control code documentation in particular, a successful document often meets two criteria: (1) the document covers the necessary information of the object being described (i.e. Document Coherence) and (2) the document is available on time (i.e., Document Urgency).

- **Step 2: For each identified factor, identify the sub-factors influencing the factor.** GAMP method stops at level 2 factors (cf. Figure 1); however, this step can be further applied for sub-factors if necessary. It should be noted that the ratings in the GAMP method may cause confusion. For example, a **Risk class** rating of 3 and low **Detectability** result in low **Risk Priority**, while a **Risk class** rating as 1 and low **Detectability** result in high **Risk Priority**. This paper does not aim to address the above issue but just proposes some modifications regarding the ratings. Still, the offset mechanism of the GAMP method is followed by grouping related (sub-)factors into classes, but the classes' rating is based on Roman numbers. Besides the three-level rating of the GAMP method, the rating of (sub-)factors is based on Arabic numerals. Thus, all low Arabic number ratings indicate low risk and vice versa. In addition, the ranges of the (sub-)factors' rating are more flexible since some (sub-)factors might need only two or more than three levels.

- **Step 3: Snowballing to extend the factor classification.** Related aspects of identified sub-factor are explored. For example, different **Change Type** of control code results in different change of comment which has different quality (i.e. **Comment Quality**). Thus, standards of comment (i.e. **Comment Compliance**) are included in the factor classification.

- **Step 4: Propose calculation rules for (sub-)factors.** Once sub-factors are selected, the calculation rules are proposed.

- **Step 5: Conduct multi-stage assessments.** The GAMP method starts with two sub-factors at level 2 (i.e. **Severity** and **Probability**) to assess their corresponding factor at level 1 (i.e. **Risk class**). Thus, the assessment might start from the lowest-level sub-factors and gradually move up the factor hierarchy. However, as mentioned, to enable flexibility, the calculation process and operators are free of choice.

- **Step 6: Visualise and interpret the results for further action.** The value of RPI4DD is reviewed to indicate the areas if the document is insufficient, incomplete, or outdated, which indicates documentation debt.

In the next section, the results of the approach are presented.

4.3 Factor classification of RPI4DD formulation

From the approach presented in Section 4.2, a basic structure to calculate RPI4DD is derived with two main influencing factors (cf. equation (3)). The **Document Urgency** factor
Example of aspects related to documentation
• Document Urgency
• Document Coherence

Change Type (sub-factor of Document Coherence factor) could refer to the change categories reported in [12]:
• Enhancements = 1
• Bug Fixes = 2
• New Features = 3
• New Developments = 4

Based on proven facts of critical category in [12]:

Different Change Type of code results in different changes of comment which has different quality (i.e. Comment Quality).

=> Explore standards of comment (i.e. Comment Compliance)

Legend

Start / End
Main activity
Sub-activity
Condition
Notes
Flow
Sub-activity flow

Following GAMP method, Comment Quality is offset against Comment Compliance. This calculation determines the Comment Class (I to III).

Comment Class is offset against the Change Type and results in Document Coherence (low, medium, high).

TD identification based on RPI4DD

In the following, the specification of sub-factors and corresponding rationale are described in the order presented in Figure 4.

4.3.1 Document urgency (RPI_Urgency)

RPI_Urgency is calculated based on three sub-factors: (1) Functionality, (2) Required Change On-site, and (3) Grade Of Test/Quality Of Test.

4.3.1.1 Functionality (RPI_Functionality)

Depending on the complexity of an implemented functionality, different amounts of explanatory documentation are required to enable maintainability by different persons [26]. Poor functional description availability (#72) poses a high risk since comprehensibility (readability) received the highest rank among the software properties influenced by complexity [5]. Therefore, Complexity Of Software is identified as a sub-factor of Functionality. The Overall Complexity metric

\[
RPI4DD = RPI_{Functionality} \cdot RPI_{On-site} \cdot RPI_{Test} \cdot RPI_{Coherence}
\]

(3)
proposed in [5] has proven as a reliable means to quantify different aspects of software complexity and, thus, could be used as a calculation rule for the sub-factor Complexity Of Software. As identified in [5], the list of characteristics includes:

- ProgramLength $M_1$
- Cyclomatic Complexity $M_2$
- FanIn_FanOut $M_3$
- Vocabulary Size $M_4$
- Difficulty $M_5$
- Data Structure $M_6$

It is worth noting that Cyclomatic Complexity or Data Structure may be weighted more strongly as they might have larger effects on the program complexity.

Second, the difficulty of software functionalities might vary [5]. As documentation activity is quite laborious and tedious, the documentation effort needs to be prioritised for the most difficult functionality (i.e., goal-oriented behavior [13]) of software. Therefore, Difficulty Of Functionality is included as another sub-factor of $RPI_{Functionality}$. As identified in [5, 13, 27], the list of functionalities is as follows.

- Organizational (low weight = 1).
- Messages (low weight = 1).

---

**Figure 4:** Selected factors influencing control code documentation (i.e. factor classification of RPI4DD formulation) following the systematic classification of Li et al. [15] and GAMP procedure [10].

<table>
<thead>
<tr>
<th>Sub-factor</th>
<th>Factor</th>
<th>Related source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity Of Software</td>
<td>ProgramLength</td>
<td>Ref [5, 26, 28] and expert responses (#72)</td>
</tr>
<tr>
<td></td>
<td>Cyclomatic Complexity</td>
<td>Ref [1, 5, 13, 27, 28] and expert responses (#51, #52, #72)</td>
</tr>
<tr>
<td></td>
<td>FanIn_FanOut</td>
<td>Ref [5]</td>
</tr>
<tr>
<td></td>
<td>Vocabulary Size</td>
<td>Ref [9, 30] and expert responses (#34, #77)</td>
</tr>
<tr>
<td></td>
<td>Difficulty</td>
<td>Ref [23]</td>
</tr>
<tr>
<td></td>
<td>Data Structure</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-factor</th>
<th>Factor</th>
<th>Related source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Organizational</td>
<td>Ref [1, 8] and expert responses (#72)</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>Ref [2, 5]</td>
</tr>
<tr>
<td></td>
<td>Status</td>
<td>Ref [29]</td>
</tr>
<tr>
<td></td>
<td>Error</td>
<td>Ref [12]</td>
</tr>
<tr>
<td></td>
<td>Motion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Safety-related</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-factor</th>
<th>Factor</th>
<th>Related source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Type</td>
<td>Required Change Class</td>
<td>Ref [1, 8] and expert responses (#72)</td>
</tr>
<tr>
<td></td>
<td>Change Frequency</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Change Source</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-factor</th>
<th>Factor</th>
<th>Related source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>TestDepth</td>
<td>Ref [29]</td>
</tr>
<tr>
<td></td>
<td>BranchCoverage</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-factor</th>
<th>Factor</th>
<th>Related source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Type</td>
<td>Enhancements</td>
<td>Ref [9, 30] and expert responses (#34, #77)</td>
</tr>
<tr>
<td></td>
<td>Bug Fixes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>New Features</td>
<td></td>
</tr>
<tr>
<td></td>
<td>New Developments</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-factor</th>
<th>Factor</th>
<th>Related source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment Class</td>
<td>Guidelines, checklists or coding standards</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HeaderCommentLinesOfCode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MultiLineComments</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SourceCodeCommentedRatio</td>
<td></td>
</tr>
<tr>
<td>Comment Quality</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Related source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ref [5, 26, 28]</td>
</tr>
<tr>
<td>Ref [1, 5, 13, 27, 28]</td>
</tr>
<tr>
<td>Ref [1, 8]</td>
</tr>
<tr>
<td>Ref [9, 30]</td>
</tr>
<tr>
<td>Ref [2, 5]</td>
</tr>
<tr>
<td>Ref [29]</td>
</tr>
<tr>
<td>Ref [12]</td>
</tr>
<tr>
<td>Ref [23]</td>
</tr>
</tbody>
</table>
– Status (low weight = 1).
– Error/generic/homing (medium weight = 2)
– Motion Control (high weight = 3)
– Safety-related (high weight = 3).

Motion control in aPS is highly challenging in case a production step (e.g., workpiece handling) requires the synchronisation of multiple drives performing different motion tasks (e.g., rotary or linear movements [28]) by simultaneously fulfilling hard real-time requirements [27]. According to [1], positioning tasks of the robot-like-systems components are rated as one of the most challenging tasks to implement in software. Changes in production (e.g., introduction of a new product) might require additional or more complex movement; thus, the motion control must be adapted accordingly (e.g., changing the number of involved cooperating motion axes of robot arms). The difficulty of motion control is therefore rated as high (2). Safety-related tasks are even more critical since errors or malfunctions in these software parts may cause severe damage to aPS, or – in the worst case – even to humans; thus, they are rated with 3. Here, poor automatic control code generation (#51 and #52) might pose a high risk, e.g., safety-related control code might differ from the predefined model or requirements document.

Following the calculation rules presented in [5, 28] for POU complexity, a calculation for \( RPI_{\text{Functionality}} \) is proposed as follows. As the medians \( \tilde{M}_1 \) to \( \tilde{M}_6 \) are used (cf. Section 2.2), they are “stable against outliers and provide reliable results” [5]. Therefore, the distribution of Overall Complexity \( \text{OC} \) values would be mostly symmetrical without outliers. Thus, \( RPI_{\text{Functionality}} \) is based on the mean Overall Complexity \( \text{OC} \).

\[
RPI_{\text{Functionality}} = \frac{\text{OC}_{\text{Organizational}} + \text{OC}_{\text{Messages}}}{\text{OC}_{\text{Status}} + 2 \cdot \text{OC}_{\text{Error}}} + 2 \cdot \text{OC}_{\text{Motion}} + 3 \cdot \text{OC}_{\text{Safety}}
\]  

(4)

4.3.1.2 Required change on-site (\( RPI_{\text{On-site}} \))

The sub-factors of Required Change On-site include Manufacturing Type, Change Frequency and Change Source. First, while MM companies can usually develop machines entirely in-house, PM companies often have to integrate and start up the system on-site [8]. Thus, the Manufacturing Type is identified as a sub-factor for Required Change On-site. Second, customers have different schedules (e.g., new product introductions); therefore, the remote software update intervals (i.e., Change Frequency) vary. Third, while in-house changes are often conducted without feedback from the development department (e.g., due to time pressure). Therefore, the qualification of the person conducting a change or the origin of changes (Change Source) is identified as another sub-factor. Other potential sub-factors include code-sharing strategies, acquiring software status, or company sizes (documentation availability might vary in sizes of companies, according to #72).

Regarding the Manufacturing Type, the highest document urgency is rated at PM (rating = 3) as PM has the most required change on site. As on-site changes must be quickly performed (e.g., to meet hard deadlines or to reduce the cost of suspending production), it is urgent to provide exact instructions. The rating is lower for series machinery manufacturing (rating = 1) and special-purpose machinery manufacturing (rating = 2).

Regarding Change Frequency, the rating scale is based on the frequency of software updates reported in [1]:

- <3 months = 1
- 3–6 months = 2
- 6–12 months = 3
- >12 months = 4
- No update = 5

Regarding Change Source, a rating of 1 is assigned to Changes conducted in-house (less risky due to rigorous review), and On-site changes are given a rating of 2.

The assessment on Required Change On-site is described in Figure 5. Following the idea in the GAMP method, this assessment includes two steps with three aspects. Firstly, the Manufacturing Type (severity of change) is plotted against the Change Frequency (probability that a change will occur), giving a Required Change Class (I to III). Secondly, the Required Change Class is plotted against Change Source, thus resulting in giving a degree of Required Change On-site (low, medium, high). There is an adaption in this assessment that the detectability aspect introduced in the GAMP method is replaced by Change Source. Nevertheless, the detectability concept is still included since the detectability of inadequate documentation may vary with different Change Sources, which might undergo different review processes.

Finally, the value \( RPI_{\text{On-site}} \) is determined using equation (5).

\[
RPI_{\text{On-site}} = \begin{cases} 
1 & \text{if low urgency on-site} \\
5 & \text{if medium urgency on-site} \\
10 & \text{if high urgency on-site}
\end{cases}
\]  

(5)

4.3.1.3 Grade Of Test/Quality Of Test (\( RPI_{\text{Test}} \))

In the aPS domain, the method proposed by Ulewicz et al. [29] can be followed to identify which control code snippets...
are not well tested yet. The process employed the control code coverage method to determine which control code snippets are covered by which test cases. Among the proposed metrics in [29], there is a test coverage measure on the POU hierarchy (i.e., TestDepth) and three control code coverage measures (i.e., BranchCoverage, StatementCoverage, and PathCoverage). With StatementCoverage, it is aimed to traverse all the statements (or lines) at least once. BranchCoverage aims to traverse all the control flow paths (e.g., if statements) at least once. The main goal of PathCoverage is to execute all possible control flow paths (full paths from input to output). A brief comparison of these control code coverage metrics is illustrated in Figure 6. With test cases TC1 and TC2 (cf. Figure 6a), StatementCoverage shows 100%; however, only 75% of the branches are covered (BranchCoverage). Thus, BranchCoverage is a better indicator as it shows that more test cases are required. Regarding PathCoverage, there is a case that the number of paths through the loop could be significantly large (cf. Figure 6b), infinite, or not predictable (e.g., number of loops is determined at run time). Thus, covering all possible paths from input to output is mostly not realisable in practice. Therefore, among these control code coverage metrics, BranchCoverage is recommended as it delivers more accurate results.

In practice, 80% coverage is generally accepted as good (following the Pareto principle in testing: given that 80% of code is covered, the remaining code part may require a significant effort but is not worth it). To distinguish between a poor and a moderate test quality, additional information needs to be taken into account. More precisely, the expectation for test quality depends on various company-specific boundary conditions, such as software modularity or reusability. For instance, the boundary conditions at the industrial partner in the previous study [25] are as follows: the company has a high-modularised structure of its control code and a high degree of reuse by applying library modules and templates (about 75% of a machine’s PLC project consist of reused control code). The other 25% of the machine’s PLC project is newly developed machine- or customer-specific control code, e.g., data exchange between the reused library modules or implementation of the machine-specific process logic. These newly developed, machine-specific parts are the error-prone parts of the PLC project, which require thorough testing. Generally, the company’s engineers know which requirements target these new code parts, and due to the company’s mature code structure, they can locate the code parts in the PLC project. Consequently, the defined tests usually aim at targeting specifically the requirements linked to these critical code parts. Again, following the Pareto principle, at least 80% of the 25% new machine-specific code needs to be covered by tests. Since the 80% of the 25% new machine-specific code is about 20% of the total code, the company-specific threshold for moderate test quality is set to 20% of the entire control code in the considered PLC project. The threshold of 20% for “moderate” is set given the assumption that test cases for the machine-specific part are selected and that the remaining code is already well-tested and thus assumed to be less error-prone. Therefore, a proposal to assess the test quality with two metrics, TestDepth and BranchCoverage, is shown in equation (6).
Finally, the value $RPI_{\text{Test}}$ is determined using equation (7).

$$RPI_{\text{Test}} = \begin{cases} 1 & \text{if good test quality} \\ 5 & \text{if moderate test quality} \\ 10 & \text{if poor test quality} \end{cases}$$

As this paper’s scope focuses on control code documentation, there is no claim for completeness regarding test metrics. Nevertheless, many test metrics are available in the literature that can be integrated into the $RPI_{\text{Test}}$ calculation.

### 4.3.2 Document Coherence ($RPI_{\text{Coherence}}$)

The Document Coherence factor is assessed based on the three sub-factors Comment Compliance, Comment Quality and Change Type. Firstly, practice shows that implementation of the standards could reduce errors. Violation of coding standards (e.g., MISRA) may result in TD [30]. A lack of exact instructions (#77) or a lack of coding guidelines in general (#34) might pose a risk, e.g., poor quality or standardised code. Therefore, Comment Compliance is identified as a sub-factor, which is categorised into the Document Coherence factor. Secondly, the coherence of control code documentation and corresponding implementation may influence the software’s maintainability. Insufficient Comment Quality (e.g., code comments do not describe the implementation properly) may hinder on comprehensibility (readability) of the control code (e.g., causing confusion for the maintenance staff). Thirdly, different change categories (i.e., Enhancements, Bug Fixes, New Features, and New Developments) could influence the software maturity value, according to [12]. Thus, Change Type is identified as a factor within the Document Coherence factor.

Regarding Comment Compliance, the rating is based on results from checking the control code documentation with MISRA guidelines. MISRA rules or directives for comments include:
Rule 3.1 The character sequences /* and // shall not be used within a comment.

Rule 3.2 Line-splicing shall not be used in // comments.

Rule 20.1 Include directives should only be preceded by preprocessor directives or comments.

Dir 4.4 Sections of code should not be “commented out”

The details of the rating scale for Comment Compliance are presented in Table 5. Among the available guidelines, MISRA is proposed as its scope focuses on coding standards. As an alternative or a broader scope, one could refer to other guidelines or standards such as PLCopen, ISO 26262 or ISO 17961.

Regarding Comment Quality (CQ) rating, it is assumed that a high amount of comments is beneficial to enhance the software’s understandability, and, thus, reduces the risk of documentation debt. The metrics HeaderCommentLinesOfCode (size of comment header), MultiLineComments (amount of comments wrapping over multiple lines), and SourceCodeCommentedRatio (density of comments) proposed in [23] could be employed to quantify the amount of control code comments (cf. equation (8)).

\[
CQ = \begin{cases} 
1 & \text{if } \text{SourceCodeCommentedRatio} > 0 \land \text{MultiLineComments} > 0 \land \text{HeaderCommentLinesOfCode} > 0 \\
2 & \text{if } \text{SourceCodeCommentedRatio} > 0 \land (\text{MultiLineComments} = 0 \lor \text{HeaderCommentLinesOfCode} = 0) \\
3 & \text{if } \text{SourceCodeCommentedRatio} = 0 
\end{cases} \quad (8)
\]

Regarding Change Type, the scale refers to the change categories reported in [12]. The details of the rating scale are presented in Table 6.

The Document Coherence assessment method is described in Figure 7. Following the idea of the GAMP method, firstly, the Comment Quality (determined using equation (8)) is offset against the Comment Compliance (determined using the rating scale in Table 5). This calculation determines the degree of excellence of comment, i.e., Comment Class (I to III). Secondly, the Comment Class is offset against the degree of excellence of control code (represented by Change Type, which influences the software maturity value). It results in Document Coherence (low, medium, high). The Change Type is determined using the rating scale in Table 6. It could be observed that the three aspects defined in GAMP (i.e., severity, probability, and detectability) are not used but replaced by the three new aspects in the context of coherent assessment.

Finally, the \( \text{RPI}_{\text{Coherence}} \) is determined with equation (9).

\[
\text{RPI}_{\text{Coherence}} = \begin{cases} 
1 & \text{if } \text{high document coherence} \\
5 & \text{if } \text{medium document coherence} \\
10 & \text{if } \text{low document coherence} 
\end{cases} \quad (9)
\]

4.4 Proof of concept of risk prioritisation indicator of documentation debt

To illustrate the applicability, RPI4DD is in the following determined for two scenarios of the extended Pick and Place Unit (xPPU) [8], i.e., a lab-sized demonstrator that stamps, transports, and sorts work pieces with different color, weight, and material (cf. Figure 8). The work pieces arrive at the stack, where they are picked up by the crane. Depending on the work piece type, they are either further processed to the stamp or directly transported with the conveyor to be sorted into the respective ramp.

In the following, RPI4DD is calculated for two typical functionalities in aPS emulated for the xPPU, i.e.:

- \( F_{\text{Stamp}}: \text{Stamping work pieces} \): This scenario refers to the code parts controlling a typical functionality of the system during regular production, i.e., in the case of the xPPU, the stamping of work pieces at the crane before they are sorted.

- \( F_{\text{Restart}}: \text{Restart operation after emergency stop} \): The recovery of the system after an emergency stop to take up regular production is a highly challenging task in industrial practice, e.g., due to complex resynchronisation of drives, which is why the scenario is considered on a small scale for the xPPU.

Table 7 follows a practical risk assessment template used to evaluate risks during the development of industrial aPS. The table template can serve different views to different stakeholders. The individual sub-factors are intended to be initially determined by technicians on a fine-grained level resulting in precise numbers for the different factors (e.g., results from complexity metrics). Next, the fine-grained values are clustered and categorised into a coarse-grained system (e.g., following a three-level categorisation into low,
Table 5: Rating scale for Comment Compliance with guidelines or standards.

<table>
<thead>
<tr>
<th>Rating</th>
<th>Description</th>
<th>Comment Compliance from checking with guidelines (e.g., MISRA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Noncritical</td>
<td>Fully compliance (i.e., no error or warning messages)</td>
</tr>
<tr>
<td>2</td>
<td>Minor</td>
<td>Partially compliance (i.e., violations only include warning messages)</td>
</tr>
<tr>
<td>3</td>
<td>Critical</td>
<td>Noncompliance (i.e., violations include error messages)</td>
</tr>
</tbody>
</table>

Table 6: Rating scale for Change Type based on scope and criticality.

<table>
<thead>
<tr>
<th>Rating</th>
<th>Description</th>
<th>Rationale according to [12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Enhancements</td>
<td>[...] represent the least critical category [...]</td>
</tr>
<tr>
<td>2</td>
<td>Bug fixes</td>
<td>[...] represent a more critical [...] than enhancements [...]</td>
</tr>
<tr>
<td>3</td>
<td>New features</td>
<td>[...] significantly more critical [...]</td>
</tr>
<tr>
<td>4</td>
<td>New developments</td>
<td>[...] the most extensive change category [...]</td>
</tr>
</tbody>
</table>

The bold terms highlight the exact phases extracted from ref. [12] explaining the rationale for the corresponding ratings in column 1.

Figure 7: Document Coherence assessment according to the GAMP method [10] (a: Comment Class calculation; b: Document Coherence calculation).

Figure 8: Extended Pick and Place Unit [8] to apply the risk priority indicator RPI4DD.

In the following, the sub-factors are determined for both scenarios to derive RPI4DD for the software parts implementing the respective functionalities.

4.4.1 Calculation of document urgency (RPI\textsubscript{Urgency})

Regarding the Functionality, F\_Restart might involve human intervention and multiple safety-related tasks; therefore, it is assumed that documentation for F\_Restart is more critical than documentation for F\_Stamp. Particularly, the Complexity Of Software and Difficulty Of Functionality of POUs in F\_Restart is assumed to be higher than F\_Stamp’s in general; thus, \textit{RPI}\textsubscript{Functionality} of F\_Restart appears to be higher than \textit{RPI}\textsubscript{Functionality} of F\_Stamp, according to \textit{RPI}\textsubscript{Functionality} calculation rules in Section 4.3. Hence, \textit{RPI}\textsubscript{Functionality} of F\_Stamp and F\_Restart are rated with 5 and 10, respectively. As \textit{RPI}\textsubscript{Functionality} calculation rules follow the mechanism of [5], they are fully automatable (cf. requirement R3).
Table 7: A (fictive) example on RPI4DD calculation on two xPPU functionalities Stamping work pieces (F_Stamp) and Restart operation after emergency stop (F_Restart).

<table>
<thead>
<tr>
<th>Functionality</th>
<th>F_Stamp</th>
<th>F_Restart</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Document Urgency</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functionality RPI$_{\text{Functionality}}$</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Required Change On-site RPI$_{\text{On-site}}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grade Of Test/Quality Of Test RPI$_{\text{Test}}$</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>RPI$<em>{\text{Urgency}}$ (RPI$</em>{\text{Functionality}}$ x RPI$<em>{\text{On-site}}$ x RPI$</em>{\text{Test}}$)</td>
<td>50</td>
<td>250</td>
</tr>
<tr>
<td><strong>Document Coherence (*)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comment Compliance</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Comment Quality</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Comment Class (Comment Compliance x Comment Quality)</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Change Type</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>RPI$_{\text{Coherence}}$ (Comment Class x Change Type)</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td><strong>RPI4DD (RPI$<em>{\text{Urgency}}$ x RPI$</em>{\text{Coherence}}$)</strong></td>
<td>50</td>
<td>2500</td>
</tr>
</tbody>
</table>

(*) cf. Fig. 7 and equation (9) for calculation rules of Document Coherence

(**) Risk Priority Risk of Documentation is assumed

- Low if RPI4DD ≤ 625 (5x5x5x5)
- Medium if 625 < RPI4DD < 1250 (10x5x5x5)
- High if RPI4DD ≥ 1250

Regarding the Required Change On-site, due to the involvement of human intervention and multiple tasks, it is assumed that F_Restart has a larger scope and requires more on-site changes than F_Stamp. In particular, due to multiple-machine involvement, the F_Restart would mostly need to be integrated on-site by PM, while F_Stamp can be seen as a series machine which could be developed in-house by MM. Thus, the Manufacturing Type of F_Restart tends to be larger than F_Stamp. The same tendency applies to Change Frequency and Change Source. Thus, RPI$_{\text{On-site}}$ of F_Restart appears to be higher than RPI$_{\text{On-site}}$ of F_Stamp, according to the RPI$_{\text{On-site}}$ calculation rules listed in Section 4.3. Hence, the Required Change On-site of F_Stamp and F_Restart are rated with 1 and 5, respectively. It should be noted that the calculation structure is extensible (cf. requirement R1), which allows the integration of other sub-factors. For instance, Facility Availability sub-factor can be included to present the readiness of required equipment to conduct the changes. In addition, the listed rating scales of Manufacturing Type, Change Frequency and Change Source can be modified (cf. requirement R2). For instance, a new item Changes conducted both in-house and on-site can be included in Change Source for a large project.

Regarding the Grade Of Test/Quality Of Test, per larger scope assumption, F_Restart has larger impact than F_Stamp in the whole aPS; therefore, it is assumed that more resources are allocated on testing and documentation for F_Restart. It is because a defective stamping machine could be quickly fixed or replaced to resume operation, while a suspension of the whole production to fix an issue in F_Restart might result in a high cost to both aPS manufacturer and customers. With larger test efforts spent, Test-Depth and BranchCoverage of F_Restart are assumed to be larger than 80% (good test quality) and F_Stamp’s are below...
80% (moderate). Thus, $RPI_{Test}$ of $F_{Restart}$ appears to be less than $RPI_{Test}$ of $F_{Stamp}$, according to the equations (6) and (7). Hence, $RPI_{Test}$ of $F_{Stamp}$ and $F_{Restart}$ are rated as 10 and 5, respectively. As results of the test metrics are provided by test management tools [29], the Grade Of Test/Quality Of Test calculation can be fully automatable (cf. requirement R3).

The $RPI_{Urgency}$ is determined by multiplying the $RPI_{Functionality}$, $RPI_{On-site}$ and $RPI_{Test}$, according to equation (3).

### 4.4.2 Calculation of Document Coherence ($RPI_{Coherence}$)

Regarding the Comment Compliance, the scopes of both functionalities are not trivial; therefore, it is assumed that there are error messages associated with the Comment Compliance rules listed in Section 4.3. For the fictive examples focused for the $RPI$ calculation for the xPPU, it is assumed that both contain comments that are non-compliant with the MISRA comment guidelines, e.g., in the implementation of the crane (cf. Figure 9). These non-compliant violations are critical for both functionalities $F_{Stamp}$ and $F_{Restart}$, thus are rated with 3 for Comment Compliance. It should be noted that there are various configurable tools available to check compliance with MISRA rules, enabling automated assessment for this criterion (cf. requirement R3).

Regarding Comment Quality, as aforementioned, $F_{Restart}$ scope is larger and relates to plant manufacturing which mostly requires on-site changes, while $F_{Stamp}$ scope mostly relates to series machine manufacturing which could be mainly developed in-house. Therefore, it is assumed that the three metrics HeaderCommentLinesOfCode, MultiLineComments, and SourceCodeCommentedRatio deliver better results at $F_{Stamp}$ than $F_{Restart}$. For instance, some header comments in $F_{Restart}$ on-site changes might be neglected due to high time pressure and short deadline, resulting in $HeaderCommentLinesOfCode = 0$. Thus, the Comment Quality of $F_{Restart}$ appears to be larger than $F_{Stamp}$, according to calculation rules in equation (8). Hence, the Comment Quality of $F_{Stamp}$ and $F_{Restart}$ are rated as 1 and 2, respectively.

The Comment Class is determined by offsetting the Comment Compliance against Comment Quality (cf. detailed calculation rules in Figure 7a).

Regarding the Change Type, a change at $F_{Restart}$ often requires a configuration of multiple tasks at multiple components; therefore, it is assumed that a change at $F_{Restart}$ is generally larger than a change at $F_{Stamp}$ (e.g., mainly New Features at $F_{Restart}$ and mainly Enhancements at $F_{Stamp}$). Thus, according to the rating scale in Table 6, Change Type of $F_{Stamp}$ and $F_{Restart}$ are rated with 1 and 3, respectively.

Altogether, the RPI4DD is determined by multiplying the $RPI_{Urgency}$ and $RPI_{Coherence}$, according to equation (3). The RPI4DD results indicate that the risk of documentation debt at $F_{Restart}$ (2500) is higher than at $F_{Stamp}$ (50). Thus, document improvements or actions to prevent damage from documentation TD related to $F_{Restart}$ should be planned for corresponding module developers or application engineers. When transferring the results received for the xPPU to real-world production systems, companies may benefit by RPI4DD as a quantitative indicator for different stakeholders (e.g., from management or software development) to prioritize starting points for reducing the amount of documentation debt and, thus, avoid high long-term cost.

### 4.5 Summary of RPI4DD calculation and requirement fulfilment

The calculation for RPI4DD is summarised as follows (cf. equation (10)). The range of each factor $RPI_{Functionality}$, $RPI_{On-site}$, $RPI_{Test}$ and $RPI_{Coherence}$ is (0, 10] with an assumption that all operands are equally critical for TD. This proposal employs the widely-used scale up to 10 for these factors to simplify the calculations. To scale the $RPI$ to another range, one can use weights

![Figure 9: A (fictive) example of non-compliant comment based on MISRA rule 3.1 example [9] and xPPU [8].](image-url)
(cf. $RPI_{\text{Functionality}}$ calculation) or offset the factors (cf. $RPI_{\text{On-site}}$ calculation). It could be observed that the range of $RPI_{\text{Urgency}}$ (multiplication of $RPI_{\text{Functionality}}$, $RPI_{\text{On-site}}$, and $RPI_{\text{Test}}$) $(0, 1000]$ is larger than $RPI_{\text{Coherence}}$ range $(0, 10]$. It is due to the criticality of documentation being considered more important than the coherence of documentation. For example, if a commissioner has to make changes at the construction site at the customer’s site, it is crucial that documentation is available (urgent), but it is not a priority whether the comments fulfill the MISRA guidelines (coherent). It is worth noting that the factor Grade Of Test/Quality Of Test could be optional. $RPI_{\text{Test}}$ is only relevant in case changes are conducted in-house (e.g., by module developer or application engineer), and test results are available for assessment. $RPI_{\text{Test}}$ acts as a considered issue to tweak RPI4DD. Square brackets denote the optionality of $RPI_{\text{Test}}$.

$$RPI4DD \cong RPI_{\text{Urgency}} \cdot RPI_{\text{Coherence}}$$

$$\cong (RPI_{\text{Functionality}} \cdot RPI_{\text{On-site}} \cdot [RPI_{\text{Test}}]) \cdot RPI_{\text{Coherence}}$$

(10)

The RPI4DD approach is achieved with a set of selected factors to indicate control code documentation debt in aPS (cf. Figure 4), which is extensible (R1), flexible (R2), and automatable (R3), therefore, feasible for industrial large-scale software projects, which often involve hundreds of POU's. The extensibility and flexibility characteristics presented in the Required Change On-site assessment (cf. Section 4.4) can generally be applied to assessments of other aspects (cf. Facility Availability and Manufacturing Type examples in Required Change On-site in Section 4.4). New sub-factors can be systematically included in the proposed hierarchy (cf. Figure 4). All proposed rating scales or calculation rules are adjustable. Thus, R1 (extensible) and R2 (flexible) are satisfied. R3 (automatable) is evaluated as partially fulfilled since all presented calculation rules can be automated except Change Source, which might require a manual assessment. Nevertheless, with trivial effort, semi-automatic assessment methods can be developed to classify Change Source (e.g., by a configuration file).

To summarise, the integration of RPI4DD into the risk management process of machine and plant manufacturing companies yields high potential to support practitioners in identifying and analysing the risks associated with documentation debt in technical systems by providing an intuitive quantitative indicator based on established risk assessment approaches. Therefore, earlier reaction or risk-reduction measures can be developed to enhance the system quality and drastically reduce costs by preventing errors and mal-functions caused by a lack of documentation in early phases.

5 Conclusion and outlook

The current state of practice regarding documentation in aPS manufacturing companies clearly showed a lack of documentation in industrial development, especially in control software, causing a high risk of maintainability issues in later lifecycle phases and thus, increased cost. In particular, the evaluation of expert surveys indicated low document accessibility, especially in early engineering phases, poor automatically generated engineering artefacts, inadequate formulation of guidelines, and highly dependent on in-house guidelines in MM and PM. To address this issue, a Risk Prioritisation Indicator of Documentation Debt (RPI4DD) is proposed to quantify the risk of documentation debt in control software. Related work addressing the quality of control code documentation is still scarce and primarily considers source code alone. To the best of the authors’ knowledge, this is the first study to transfer the Risk Priority concept in the GAMP method to automation software and its boundary conditions. The RPI4DD approach considers not only internal software factors (e.g., complexity or functionalities) but also includes external influencing factors, such as required on-site changes or tests. Therefore, compared to existing work, the proposed approach could provide a broader view on control code documentation since aPS is developed in a multidisciplinary environment. Besides documentation, the proposed approach could be generally applied to other aspects of software quality (e.g., testing) or could be further applied to other disciplines.

The applicability of the approach is evaluated on a lab-sized demonstrator. Based on the risk priorities obtained, follow-up documentation activities can be determined. No action is required in case the risk priority is low, e.g., F_Snap in the proof of concept presented in Section 4.4. If the risk is high, e.g., F_Restart, the factors contributing most to the risk must be analysed to plan for additional documentation tasks, e.g., to review and note changes of F_Restart at commissioning since F_Restart would mostly need to be integrated on-site by PM – as aforementioned. If the outcome is a medium level of risk, it is necessary to check if current documentation is sufficient for staff working with the control code.

The presented factors covering an initial basis of influences need to be adapted or enlarged (e.g., quality of code comments is not yet covered). The evaluated use case is limited to the scope of a lab-sized demonstrator consisting of general functionalities. There may be unexpected
company-specific software structure and documentation. More concrete factors might be required since the code size of industrial applications may vary from a hundred to several thousand lines of code per POU [12]. In future work, it is therefore planned to evaluate RPI4DD in industrial settings to enhance the factor hierarchy and calculation rules. First, the workflow with involved stakeholders and communication interfaces between departments would need a study to collect the required information for RPI4DD calculation in practice. Second, control code documentation refers not only to code comments but also to manuals or architect documents; thus, these documentation artefacts should be included in future studies to further develop the factor classification for the extensible RPI4DD. Furthermore, new tools can be integrated or designed to enable automatic assessments for the identified factors to promote the RPI4DD concept to a framework applicable to industrial development workflows.

**Acknowledgements:** We would like to thank Kathrin Land and Juliane Fischer for the support in the research. We would also thank the Vietnam International Education Development and Vietnamese-German University for granting Quang Huan Dong a scholarship under Project 911 – “Training lecturers of Doctor’s Degree for universities and colleges for the 2010–2020 period” (Decision No. 771/QD BGDDT dated 14/03/2017).

**Author contributions:** All the authors have accepted responsibility for the entire content of this submitted manuscript and approved submission.

**Research funding:** None declared.

**Conflict of interest statement:** The authors declare no conflicts of interest regarding this article.

**Appendix**

This section provides excerpt of the questionnaire used in this study. The full questionnaire is available online [24].

- #34: What guidelines/checklists does your company use to ensure high quality and traceability software?
- #51: What types of planned reuse do you use for the development of open-/closed-loop or drive software?
- #52: From which tools/models is code generated?
- #65: Which sales markets is your company active in?
- #66: How many employees work in your company?
- #69: Would you classify your company more as a machine or plant engineering company?
- #72: When are the functional descriptions available in the respective department?
- #77: Does your company use internal guidelines for the respective discipline that serve to achieve results of the highest possible quality and traceability?

**References**


### Bionotes

**Quang Huan Dong**
Institute of Automation and Information Systems, Department of Mechanical Engineering, TUM School of Engineering and Design, Technical University of Munich, Munich, Germany
Vietnamese-German University, Thu Dau Mot City, Vietnam
huan.dong@tum.de

Quang Huan Dong is a PhD candidate at Institute of Automation and Information Systems at Technical University of Munich. His research interest is technical debt in automated production systems.

**Eva-Maria Neumann**
Institute of Automation and Information Systems, Department of Mechanical Engineering, TUM School of Engineering and Design, Technical University of Munich, Munich, Germany
eva-maria.neumann@tum.de

Eva-Maria Neumann received an M.Sc. in Mechanical Engineering from Technical University of Munich (TUM), Munich, Germany in 2018. She is currently pursuing a Ph.D. at the Institute of Automation and Information Systems at TUM. Her main research interests are static code analysis and metrics to quantify control software quality and the function-oriented design of modular control software architectures to enhance its reusability.

**Birgit Vogel-Heuser**
Institute of Automation and Information Systems, Department of Mechanical Engineering, TUM School of Engineering and Design, Technical University of Munich, Munich, Germany
Core Member of MDSI and Lead of Sector Work of MIRMI, Technical University of Munich, Munich, Germany
Munich Institute of Integrated Materials, Energy and Process Engineering, Technical University of Munich, Garching, Germany
vogel-heuser@tum.de

Prof. Dr.-Ing. Birgit Vogel-Heuser received a Diploma degree in Electrical Engineering and a Ph. D. degree in Mechanical Engineering from RWTH Aachen. Since 2009, she is a full professor and director of the Institute of Automation and Information Systems at the Technical University of Munich (TUM). Her current research focuses on systems and software engineering. She is member of the acatech (German National Academy of Science and Engineering), editor of IEEE T-ASE and member of the science board of MIRMI at TUM.