Beyond Code: Is There a Difference Between Comments in Visual and Textual Languages?

Alexander Boll*, Pooja Rani, Alexander Schultheiß, Timo Kehrer

Abstract

Code comments are crucial for program comprehension and maintenance. To better understand the nature and content of comments, previous work proposed taxonomies of comment information for textual languages, notably classical programming languages. However, paradigms such as model-driven or model-based engineering often promote the use of visual languages, to which existing taxonomies are not directly applicable. Taking MATLAB/Simulink as a representative of a sophisticated and widely used modeling environment, we extend a multi-language comment taxonomy onto new (visual) comment types and two new languages: Simulink and MATLAB. Furthermore, we outline Simulink commenting practices and compare them to textual languages. We analyze 259,267 comments from 9,095 Simulink models and 17,792 MATLAB scripts. We identify the comment types, their usage frequency, classify comment information, and analyze their correlations with model metrics. We manually analyze 757 comments to extend the taxonomy. We also analyze commenting guidelines and developer adherence to them. Our extended taxonomy, SCoT (Simulink Comment Taxonomy), contains 25 categories. We find that Simulink comments, although often duplicated, are used at all model hierarchy levels. Of all comment types, Annotations are used most often; Notes scarcely. Our results indicate that Simulink developers, instead of extending comments, add new ones, and rarely follow commenting guidelines. Overall, we find Simulink comment information comparable to textual languages, which highlights commenting practice similarity across languages.

Keywords: documentation, graphical, diagram, knowledge-transfer, Simulink, model-driven engineering, comment clones, taxonomy

1. Introduction

Code comments (hereinafter comments) are crucial in helping developers understand, maintain, extend source code [12, 56, 57], and find locations of interest in the source code [50]. High-quality comments, therefore, have a high impact on lowering the development cost and improving the quality of software [41]. Given the importance of comments, researchers focused on many aspects of them, *e.g.*, automatically assessing comment quality [38], comment completion [59], comment generation [16, 19], to name only a few. Recently, researchers explored the contents of class comments and categorized the various information in them [34, 40, 58]. Building on this, Rani *et al.* [39] formulated a taxonomy of comment information, called Class Comment Type Model (CCTM), containing types such as summaries, warnings, recommendations, licensing information, *etc.* A complete taxonomy of comment types is needed for further automation of tools handling comments.

Prior categorization efforts, however, were done on textual class comments of object-oriented general-purpose languages (*i.e.*, Python, Java, and Smalltalk) only. On the contrary, little is known about commenting practices in language environments using visual paradigms, such as Simulink [20, 33]. In particular, the classification taxonomy from textual languages cannot be directly transferred. Apart from the different paradigms, Simulink has several ways to comment models, and the possibilities are more diverse than purely textual comments (see Section 2.2). Furthermore, Simulink models are often designed by non-software engineers [1]. Such domain experts may employ a unique commenting culture when compared with more "classical" software engineers.

Choosing Simulink as one *representative* of visual languages – Simulink is a mature software which is widely studied and employed in several key industries [11, 14, 26, 53] – our overall goal is twofold. We first aim at getting a better understanding of commenting practice in Simulink, before a comparison to textual languages shall help us to build a bridge for transferring existing knowledge from textual to visual programming languages. To that end, we study Simulink comments and develop a classification taxonomy for them, generalizing prior work to a visual language and its more diverse types of comments. There-

*Corresponding author
Email addresses: alexander.boll@inf.unibe.ch (Alexander Boll), rani@ifi.uzh.ch (Pooja Rani), AlexanderSchultheiss@pm.me (Alexander Schultheiß), timo.kehrer@unibe.ch (Timo Kehrer)

upon, we compare major characteristics of Simulink comments with those in the textual programming languages Python, Java, and Smalltalk.

In our study, we first extract a collection of Simulink comments from a large set of open-source Simulink projects [46]. Then, we study how Simulink projects are commented, which comment features are used, where comments are present in the model and for what purpose they are used. We manually classify a sample of our collection according to the existing CCTM taxonomy [39], and extend it to make it suitable for Simulink comments, yielding the Simulink Comment Taxonomy (SCoT). We also investigate, whether model size, age, or complexity correlate with a model's commenting effort. As Simulink projects often feature MATLAB code, we include the projects' MATLAB code in our investigations where appropriate. Then, we compare the commenting practices of Simulink and MATLAB with the practices of the previously studied languages [39], to gauge differences and similarities between them. Finally, we gather existing guidelines on MATLAB and Simulink and explore whether developers follow them.

The main findings of our study are as follows. The Simulink comment types are used in widely varying amounts, with Annotations being the most frequent comment type, while Notes are rarely used. Simulink comments are distributed evenly across all hierarchy depths, apart from the top levels, where developers clearly put in the most commenting effort. We found that size and complexity of a model correlate with the number of comments and amount of total comments of a model, but they do not correlate with the length of individual comments. This indicates that, as a model grows, developers do not add to existing comments, but create new comments instead. This underlines previous observations that Simulink comments, once created, hardly get revised [20] and also supports the claim that Simulink documentation becomes "rotten" [33]. Without adapting comments to an evolving model, developers risk that comments become out of sync with the model – which is a well-known concern from other programming languages [38]. We also found that Simulink and MATLAB comment information is highly similar in quality and quantity to previously studied comment information in Java, Python, and Smalltalk. The comments of all these languages cover mostly the same categories of our taxonomy, and these categories also show a similar distribution in all of them. This implies that, information-wise, the commenting cultures in Simulink and MATLAB are not much different from the textual languages Java, Python, and Smalltalk. We view this as an indicator that our extended taxonomy SCoT can be employed in the categorization of other programming languages. Similarly, we expect knowledge-transfer, regarding comments, between textual languages and visual languages and vice versa to be possible. Further, we believe that many of our conclusions generalize beyond Simulink to other languages and their tools. While analyzing the commenting guidelines of Simulink and MATLAB, we found only three, which developers rarely followed.

We summarize our contributions as follows:

- a qualitative and quantitative overview of Simulink commenting practices in a large and diverse set of open source projects and models;
- an empirically validated taxonomy, named SCoT (Simulink Comment Taxonomy), classifying the information of Simulink and MATLAB comments, also applicable for other languages;
- a comparison of Simulink and MATLAB comments to previously studied languages;
- a publicly available dataset of extracted comments and classified comments in the replication package,[1] as well as all scripts used in this work.

2. Background

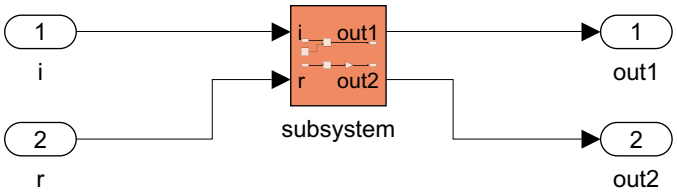
2.1. Simulink

Simulink is a visual programming language developed by MathWorks.[2] Simulink offers a modeling environment for the simulation and analysis of graphical block-oriented models of multi-domain dynamical systems. It offers a high versatility through its many toolboxes for different scenarios and domains (*e.g.*, from theoretical simulation[3] to control of tangible systems,[4] in as different domains as solar power grids [11] to automotive [53]). Simulink is a widely used modeling language for industrial-scale cyber-physical systems [14, 26] and is widely studied by researchers [8].

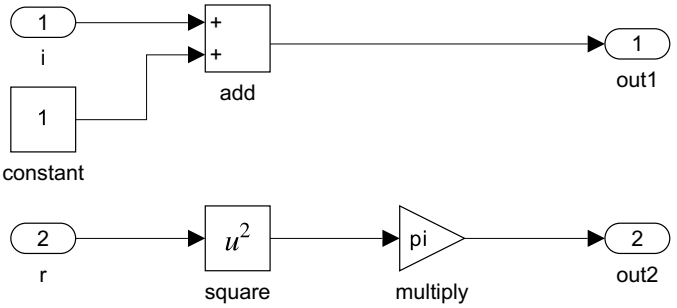
A Simulink model is a data flow graph with vertices and edges. While the edges are represented as signal lines, the vertices are different kinds of blocks. Figure 1 shows two views of an example model with its blocks connected by signal lines. Each block of a Simulink model transforms its input signals into output signals, giving a data flow-oriented model. A signal's arrowhead next to a block signifies an input; the side without an arrowhead is an output of that block.

To manage the size and complexity of a large model, it can be divided hierarchically into subsystems. Each subsystem can contain further blocks, lines, and other subsystems, recursively. Simulink then shows the *view* of the model by only presenting blocks of the currently selected subsystem and hiding blocks nested in other subsystems. The model in Figure 1 has two views: the outer view with its subsystem highlighted in apricot (Figure 1a) and the view from inside the subsystem (Figure 1b).

[1] https://doi.org/10.6084%2Fm9.figshare.24631350
[2] https://www.mathworks.com/
[3] https://www.mathworks.com/help/mpc/ug/control-of-an-inverted-pendulum-on-a-cart.html
[4] https://www.mathworks.com/help/aeroblks/quadcopter-project.html



(a) The root subsystem view of the model. A subsystem is shown in apricot, while its implementation content is hidden. The implementation content (*c.f.* Figure 1b) is only hinted at on the subsystem symbol.



(b) The view from inside the subsystem reveals the detailed implementation through all its blocks and signals.

Figure 1: Two views of an exemplary model. The model computes the functions $out_1 = i + 1$ and $out_2 = \pi r^2$. The implementation of the functions is accessible and editable in the subsystem view in Figure 1b and hidden from the outside view in Figure 1a.

2.2. Simulink Comments

Some early research claimed that models do not need documentation because "models *are* documentation" and models are less ambiguous than textual documentation [4]. Today, however, the need for a model's documentation, has become clear [33].

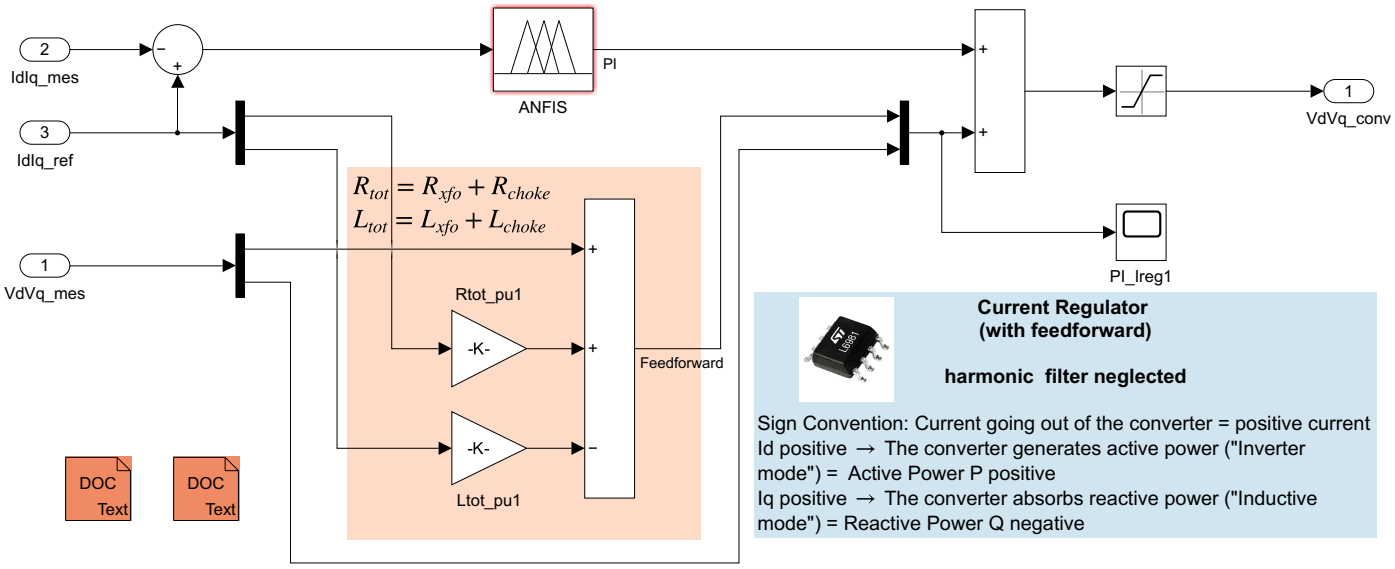
In this work, following the usual distinction between *internal* and *external* documentation [2, 30, 35], we focus on internal documentation directly integrated into the Simulink suite. Such documentation cannot get "lost" because it is in direct association with the model and will, by necessity, be as current as the model itself. Moreover, previous work on traditional programming languages has shown that developers embed various types of information in internal documentation [12, 56], which is often considered more trustworthy compared to all other sources of documentation (such as README files, user manuals, etc.) [29].

There are multiple ways of internally documenting Simulink models. At the time of writing, Simulink supports the following documentation types, which we will describe in detail below: *Model Description, Element Description, Annotation, DocBlock, and Note.* As internal documentation in textual languages is usually referred to as *code comments*, we use the term "comment" for instances of internal Simulink documentation, even though they offer much more versatility than classical comments in textual programming languages. In the sequel, we still draw a comparison to textual comment types from both a *reference* and a *usability perspective*, so that the reader can get a better understanding. Before delving into the comparisons, it is important to note that these are not meant to be scientifically rigorous analyses. Instead, they are intended to offer some preliminary insights and intuition. A comment viewed from the *reference perspective* is the part or parts of a model the comment is about. Comments in textual and visual languages are thus comparable, if they reference comparable parts of a program or model, *e.g.*, a single code line and a single model element, or the whole code file and the whole model. The *usability perspective*, on the other hand, is about how developers are able to notice, access, and edit a comment.

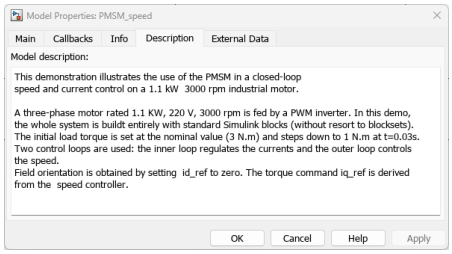
Model Description: A model can be given a single, designated textual description, which is only accessible after four mouse clicks in a popup window from Simulink's menu, and is not displayed in the main graphical view of a model (*c.f.* Figure 2b). Reference-wise, the closest analogy in classical programming languages are class comments or header comments, as there is only a single Model Description to describe the whole model. Usability-wise, the closest parallel in classical languages are README files or other external documentation, but Model Descriptions are an actual part of the Simulink model file.

Element Description: An element's description is associated to its model element (block, signal, bus). Users can describe the element, its usage, or context in more detail. An element's description text can only be accessed with two mouse clicks in a separate popup window (*c.f.* Figure 2c). Reference-wise, we view Element Descriptions as most similar to inline comments, as they refer to single model elements, which are comparable to a short line of code. Usability-wise, there is no clear parallel we know of.

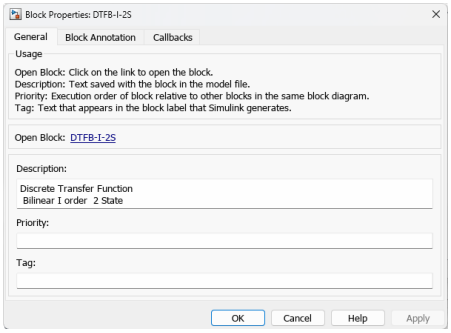
Annotation: An Annotation is a special area, placed in a model. These areas are mainly used to hold textual comments. They can also be colored and thus highlight a part of a model or even hold images. Annotations can also be linked to another model element, so the connection stays, even if the element is moved and the Annotation is not located nearby, anymore. Annotations are the only comment type of Simulink whose content is directly visible and editable in the model view. The champagne Annotation shown in Figure 2a highlights and explains a specific part of the model; the light blue Annotation gives the title of the view, further explanation, and shows various equations. There is also a small Annotation with a picture located on top of the light blue Annotation. Annotations can be used for model interaction, like holding a hyperlink to another subsystem or starting the model's simulation. Reference-wise, Annotations could be used like every type of code comment, due to their great versatility: a tiny Annotation next to



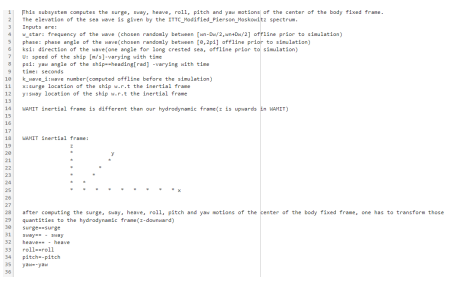
(a) A model's subsystem, featuring several comment types: DocBlocks (on the left in apricot) and Annotations (in champagne and light blue). The Annotation in champagne color highlights a specific area of the view, while the Annotation in light blue gives general information and shows a picture. Note the formatting opportunities, including LaTeX, in Annotations.



(b) A Model Description's content is shown and edited in a popup window. There is only one Model Description per model.



(c) An element (block or signal) description's content is shown and edited in a popup window.



(d) DocBlock content is shown and edited in an external editor window. One may use, *e.g.*, Microsoft Word with its features.

Figure 2: Examples of Simulink comment types.

a block like an inline comment, up to bigger Annotations describing a whole view or model, like a function or class comment. Usability-wise, they mimic all types of code comments, because of their immediacy, while additionally text formatting, pictures, and interactivity are possible.

DocBlock: A DocBlock is a special block in a Simulink model, which holds an embedded txt/html/rtf comment. As such, it can be used for longer and formatted comments. Two DocBlocks are part of the model in Figure 2a in apricot color. Although the DocBlock, as a block, is part of the graphical model view, its text can only be accessed in a separate editor window (*c.f.* Figure 2d), after a double click. Reference-wise, we view DocBlocks as most similar to function comments, because the DocBlock refers to a whole subsystem, which is comparable to a function. Usability-wise, DocBlocks work similarly to a clickable code comment hyperlink, which can be followed to some external documentation (this is sometimes used in, *e.g.*, JavaDocs), while the DocBlock and its content still is embedded in the Simulink model file itself.

Note: Simulink Notes are a mix of external and internal documentation. On the one hand, they are deeply integrated into the IDE. On the other hand, they are saved as external documentation files, only associated to a model file. A Note's textual content can be accessed with three mouse clicks in a separate editor window in the Simulink IDE, next to the model. Notes are more powerful than the other types, as they follow the model hierarchy. Depending on the current view of the model, a Note can show appropriate content only concerning this view. Thus, a single Note can be seen as a set of comments on classes or functions, reference-wise. Usability-wise, there is no clear parallel in classical languages. As our dataset lacks instances of Notes, we don't depict any in Figure 2.

2.3. MATLAB Comments

The MATLAB programming language uses textual representation for its source code. This means, the script files feature comments, similar to other textual programming languages. As Simulink models are often combined with MATLAB code in a project, we have the opportunity to study comments from bilingual projects in our work. MATLAB comments start from the %-symbol until the end of a line, or embrace comment text in between %{ and %} brackets for multi-line comments. Listing 1 shows parts of a MATLAB source code file with several comments: the first multi-line comment from lines 1 to 4 gives a title and author information. The comments in lines 6 and 9 are short inline comments.

```
1  %{
2  Logger control for <project title>
3  <Author Name>
4  %}
5
6  % Initialize Log
7  if enable_log
8    set_param('Log','logging','on');
9  else %enable_log == 0
10   set_param('Log','logging','off');
11 end
```

Listing 1: Examplary MATLAB source code showing a multi-line comment at the top and two shorter inline comments.

2.4. Simulink and MATLAB Comment Guidelines

We searched the official guidelines for High-Integrity Systems (G1),[5] and by the MathWorks Advisory Board (G2),[6] for instructions on how and when to comment in MATLAB or Simulink.

While G1 aims for "models that are complete, unambiguous, statically deterministic, robust, and verifiable", it does not provide advice on Simulink comments and gives only four guidelines regarding MATLAB comments. `himl_0001` requests to use a standardized header comment, `himl_0003` requests a comment density of 0.2 comment lines per line of code, `hisl_0038` asks for comment preservation in generated code, and `himl_0006`/`himl_0007` demands "meaningful" comments for if/else and switch statements. Note that the `else` statement of line 9 in Listing 1 is artificially commented by us.

One of the three aims of G2 is readability, which is further clarified as "improve readability of functional analysis, prevent connection mistakes, comments, etc." Still, we only found three guidelines related to Simulink documentation: `db_0140` display custom block parameters explicitly in the diagram, `db_0043` use consistent fonts and appearance settings across project, and `jc_0603` comment the model layer with a description. G2 also remarks that 'using Annotations [to group logically related parts as virtual objects] makes [the model] easier to understand'.

[5] https://www.mathworks.com/help/pdf_doc/simulink/simulink_hi_guidelines.pdf

[6] https://www.mathworks.com/help/pdf_doc/simulink/simulink_mab_guidelines.pdf

2.5. The Class Comment Type Model (CCTM)

Identifying the kinds of information embedded in code comments can support developers in various development and maintenance tasks, *e.g.*, an automatic comment classificator or updater would need a complete taxonomy of comment types. Therefore, researchers put a lot of effort in classifying code comments, building code comment taxonomies. Based on taxonomies for textual programming languages, like Java, Python, and Smalltalk [34, 40, 58], Rani *et al.* [39] presented a taxonomy of class comments, called the Class Comment Type Model (CCTM). Rani *et al.* use the standard definition of classes in object-oriented languages, *i.e.*, classes represent blueprints for building instances [55]. Class comments are expected to hold various information [32, 34, 39, 40, 58], from high-level design to low-level implementation details [32]. The CCTM can be used to classify class comments into the following higher-level categories:

Purpose: A summary of the code's intent, further explanation of how the code works, or its rationale.

Notice: An explicit notice of exceptions, warnings, deprecation, or how to use the code.

Under Development: This encompasses development notes, notice of incomplete code parts or TODO-notes. It could also be commented code, coding guidelines or recommendations for extending the code.

Style & IDE: IDE or compiler directives or a comment that visually partitions code or comments into logical sections.

Metadata: Metadata could be licensing information, ownership information, or pointers to other resources.

Discarded: A higher-level category for comments that are not further analyzed: auto generated comments, unidentifiable (noise) comments, comments in a foreign language.

The six higher-level categories are divided into 20 lower-level categories: *e.g.*, the higher-level category *Purpose* consists of the lower-level categories *Summary*, *Expand*, *Rationale*. The complete breakdown of higher-level categories into categories can be seen in Table 5.

The CCTM is based on classifying comments from a diverse set of textual languages, which is why we assume some generalizability to comments from other languages, such as MATLAB and Simulink. Also, the CCTM offers a broad spectrum with 20 categories, which makes it currently the most fine-grained taxonomy [24]. Still, it is unknown whether the taxonomy can be directly transferred

to Simulink or non-class comments in MATLAB. In this work, we use the CCTM as a first step to classify Simulink and MATLAB comments and complement it with missing categories to build our taxonomy SCoT, which is also applicable to non-class comments and visual languages like Simulink.

3. Methodology

3.1. Research Questions

The goals of this study are to explore the landscape of comments in Simulink projects, to understand how comments are used and what information they embody, and to establish a mapping of commenting practice in Simulink projects and textual programming languages. With this in mind, we design our research questions (RQs), and explain them in this section. Our focus is on Simulink models, as MATLAB is a textual language featuring comments that are similar to other textual programming languages. To put our findings for Simulink in context of those more well-understood languages, we also analyze MATLAB code from the bilingual projects of our dataset, similarly to Simulink, except for RQ 2.

***RQ 1**: How are Simulink projects documented?*

There exist various types of comments in model-based development environments, such as Simulink (see Section 2.2). Not all types of comments are expected to be used in the same frequency. We give a breakdown of the usage frequency of Simulink's comment types. As Simulink models can consist of various subsystems (or layers of subsystems), the comments can also be present in various layers of these systems. However, whether certain layers tend to be more commented than others and with which comment types is unknown. We analyze this information at all levels of depth. During our work, we found many comments to be have identical comment texts (type I comment clones [5]), in some cases hundreds of times. We refer to such comments as *duplicates* and investigate possible duplication sources of heavily duplicated comments further. Finally, we investigate, whether developers follow the guidelines we collected in Section 2.4.

With RQ 1, we aim to answer, which comment types are typically present in models, learn their basic characteristics and where they are used.

***RQ 2**: Does the amount of documentation vary in different models?*

Prior research searched for correlations between the amount of comments and other project characteristics in textual languages: *e.g.*, correlations exist between the number of comments and number of issues in the code [31], but no correlation between number of comments and number of project authors has been found [15]. However, to the best of our knowledge, it is currently unknown whether a model's age, size, and complexity and amount of comments show a correlation. With this knowledge, we can better gauge the importance of comments in big, mature, and complex models. Also, with such correlations established, comment smells [21] could be derived: developers should potentially revise the comments of strong outliers, *e.g.*, if a model grew very large but is still hardly commented.

***RQ 3**: How can the content of Simulink comments be classified?*

As comments can cover many topics, *e.g.*, summary, usage tips, licensing information *etc.* we aim to understand, what they are employed for in Simulink and MATLAB. To this end, we classify Simulink and MATLAB comments, using the CCTM taxonomy by Rani *et al.* [39] from Section 2.5. We analyze the commenting practices in terms of what information is embedded inside different comments, such as *Summary*, *Warning*, *Copyright notice*, *etc.* Please note, the CCTM is a *Class* Comment Type Model. While our MATLAB samples feature a few class comments, most are in fact inline comments. Simulink, does not even feature classes, but offers various comment types (see Section 2.2). Thus, MATLAB and Simulink comment information may fall outside the current CCTM taxonomy.

Based on this step, we propose an extended taxonomy SCoT for MATLAB and Simulink that encompasses comments from textual and visual languages.

***RQ 4**: How does Simulink documentation compare to textual programming languages?*

While the first three RQs focus on Simulink projects and exclusively on their languages Simulink and MATLAB, we also want to put these findings into context of previously studied languages. Simulink comments, with their various comment possibilities in a visual programming language, may differ significantly from textual programming languages. Depending on the results of our comparison, Simulink and MATLAB may have to be treated separately in documentation research or could be treated similarly to textual languages in some contexts.

3.2. Study Subjects and Data Collection

3.2.1. Data Set and Sample

To collect Simulink model comments and MATLAB comments, we use the SLNET set by Shrestha *et al.* [46]. Their set contains 2,833 Simulink projects, consisting of 9,095 Simulink models (we could analyze 9,033 models successfully, *i.e.*, our analysis scripts ran error-free) and 17,792 MATLAB source code files. Shrestha *et al.* curated open source Simulink projects from GitHub and MATLAB Central.[7] The projects thus represent a highly diverse data

[7] https://www.mathworks.com/MATLABcentral

set, comprising a range of tiny toy projects up to industry-like projects from various domains [7]. The SLNET set has been used in prior work for replication studies or learning about Simulink bus usage [3, 47]. We use the complete SLNET set to answer RQs 1 and 2, and have not excluded any comments, as we want to give a holistic overview of comments.

To answer RQ 3, we manually analyze a uniformly sampled subset of SLNET comments, as no automatic classifier exists for MATLAB or Simulink, yet. We thus choose the same sampling strategy as was used to create the CCTM taxonomy (see Section 2.5). We compute our sample size n, required to estimate population proportions of finite populations, according to the standard Equation (1) given by Triola *et al.* [51]:

$$n = \frac{\frac{z^2 p(1-p)}{e^2}}{1 + \frac{z^2 p(1-p)}{e^2 N}} \tag{1}$$

We choose our confidence level of 95% and thus the error $e = 0.05$, and $z = 1.96$. The value of p defaults to 0.5. We give a breakdown of sampled comments for each type of comment in the last column of Table 1. To get a better overview of the full breadth of comments, we deduplicate the SLNET comment set, before we sampled from it. After deduplication every comment has a unique comment text. This ensures that our results are not dominated by comments that are automatically generated, imported from libraries, or copy-pasted numerous times. We then use our manual analysis results of RQ 3 to answer RQ 4.

3.2.2. Extraction of Simulink Comments

We analyze each model of the SLNET set element by element to check for the presence of comments (*c.f.* Section 2.2). For each comment, we note relevant metadata, the main ones being the type of the comment (Element Description, DocBlock, *etc.*), the comment text and its length in chars, and the nesting depth in the subsystem hierarchy.

In this first step, we found only 11 instances of Simulink Notes. As there are so few of them, we investigated them manually: five of them were automatically generated, the Simulink IDE was unable to load another five, and the last one was just a test Note. Because of this, we did not sample Simulink Notes for the manual analysis of RQ 3.

In the SLNET set, we found many duplicated comments (i.e., comments with identical text). Based on Blasi *et al.* [5] and our observations, we suspect duplications coming from (i) a duplication process like copy-paste/cloning (individual comments, file duplications, or project forking), (ii) generic comments being located in multiple locations of a model (*e.g.*, copyright notice) or very short comments likely to appear more than once, due to the limited information they hold, (iii) library imports, (iv) generation by the IDE, and (v) synthetic generation. To not skew our results by heavily duplicated comments, we sample from a subset of deduplicated comments, only.

We further found that some comments stem from Mathworks' libraries or toolboxes. As they are part of the models – many toolboxes are open source projects in the SLNET set themselves – we do not exclude them from our sample. Due to the deduplication step described previously, such library comments are not overrepresented in our sample. Our sample set for manual analysis incorporates 374 Simulink comments. Table 1 gives an overview of the number of different comments, the cardinality of comment texts, and how many we sampled.

3.2.3. Extraction of MATLAB Comments

In the 2,833 projects of SLNET, there are 17,792 MATLAB source code files. In 14,642 of them, we found at least one source code comment. For the manual analysis, we sample from the deduplicated subset, which results in 383 MATLAB comments. Table 1 gives an overview of the number of MATLAB comments, the cardinality of comments, and how many we sampled.

3.2.4. Computational analysis

We extracted the Simulink comments and their metadata (see Section 3.2.2) directly from the models themselves with a MATLAB script. For this, we iterated over the whole model set, and within each model. We first collected a potential Model Description, all Annotations, and DocBlocks. Furthermore, we iterated over every model element and inspected it for a possible Element Description. We kept track of each comment and its metadata for further analysis steps.

We gathered the MATLAB comments using a Python script. We fused successive lines only containing comments to a single comment, even when the developers do not use the 'official' multi-line method of bracketing the comment between `%{` and `%}`. We did this, as the multi-line feature is not often used and developers tend to fall back to starting each line of their multi-line comment with a simple `%` symbol, even for very long comments.

All Simulink and MATLAB comments we found are gathered in `.json`-files, which we then analyzed further with Python scripts for RQs 1, 2 and 4.

3.2.5. Manual Classification Process

To answer RQ 3, we first gathered the sampled comments into a shared Google sheet[8] for a collaborative classification process. Three researchers (a postdoctoral researcher and two Ph.D. candidates) participated in the classification process. We used the same three-step classification process as was employed by Rani *et al.* [39]: we split up the samples in a way that each comment is classified by one researcher in the first step. Next, another researcher reviewed the first classification and possibly proposed changes to the classification. The original researcher then accepted or rejected the proposed changes

[8] https://www.google.com/sheets

of the reviewer. If changes were rejected (if both evaluators disagree), a third researcher reviewed the comment and gave a final verdict on the classification. During classifying and reviewing, we kept track of missing classification categories, to expand or refine the CCTM taxonomy, by new categories, we observed. For example, Simulink contains some comments that have interactive features, for which we created a new *Interactive* category. For that purpose, all three researchers discussed their disagreements in the classification/reviewing process, as they are an indicator of the potential taxonomy refinement or extension. We also noted, how many comments needed a second or third review, to gauge our inter-rating conformity. This process yielded our taxonomy SCoT, in the same way as the taxonomy CCTM (see Section 2.5) was built.

In answering RQ 4, we use our findings of RQ 3 and compare the similarity of Simulink and MATLAB commenting practice with findings of studies that used the CCTM to classify Java, Python, and Smalltalk by Rani *et al.* [39].

4. Results

In this section, we describe the results of our study structured by research question; the discussion follows in the next section.

***RQ 1**: How are Simulink projects documented?*

Table 1: The absolute number of each comment type found in the SLNET set for Simulink models and MATLAB source code files, is shown in the *comments* column. The deduplicated numbers are given in the middle column. The number of sampled comments for our manual analysis is given in the last column.

	Comment Type	*comments*	|*comments*|	*sampled*	
Simulink	Model Description	2,088	521	16	374
	Element Description	5,303	287	3	
	Annotation	91,027	11,348	348	
	DocBlock	308	129	7	
	Note	11	6	0	
MATLAB	Class Comment	472	354	3	383
	Other Comment	159,957	75,589	380	

General Measurement and Properties

We counted the total number of each comment type in Simulink models and MATLAB source code files, and depict the results in the second column of Table 1. As can be seen, Annotations make up the overwhelming majority of Simulink comments, with over 90k instances in our 9,033 Simulink models. All other comment types combined only add up to about 7.7k instances.

Almost all MATLAB comments are non-class comments. In the 552 MATLAB classes of our source code files, we found 472 of the classes to have a class comment, though.

As can be seen when comparing the absolute (*comments*) and cardinality (|*comments*|) columns of Table 1, many comment texts are duplicated in our set (*e.g.*, around 88% of the Annotation texts are duplicates). From the class comments in our set, on the other hand, only 25% are duplicates, while over half of the non-class comments are.

Comment Duplication and Duplication Reasons

To get a better understanding of comment duplicates (or clones) in Simulink and MATLAB, we present a scatter plot of duplicates in Figure 3. In the graph, the left-most comments are unique, while the right-most are heavily duplicated. The x, y-position of a marker represents that there are y different comments which are duplicated x times in our dataset. For example, more than 50k non-class comments from MATLAB are unique (dark blue marker at $x = 1, y = 54,287$), while the next marker at $x = 2, y = 11,598$ indicates that more than 10k comments of that type are duplicated exactly once; the last marker at $x = 1,524, y = 1$ represents one comment which was duplicated 1,523 times.

As can be seen in Figure 3, there are many duplicates (all comments with $x > 1$), with some comments duplicated dozens or in a few extreme cases more than a thousand times, such as Simulink Annotations or MATLAB's non-class comments. Such heavily duplicated comments are overall rare on the other hand, *i.e.*, the higher the duplication count of a comment, the lower the chance that there is another comment with a similarly high duplication count. This can also be seen at the sparsity of markers of most types for higher duplication counts. In fact, every comment type, except Element Descriptions, has more unique comments than those that have at least one duplicate. In other words: the first marker's y value of a type is higher than all the others combined.

To understand the duplication phenomenon better, we sampled the ten most duplicated comments of each category in Table 3 (represented by the right-most markers of each type in Figure 3). One can immediately see that some comments that occur most often are also highly similar, *e.g.*, the copyright notices in Model Descriptions: 1,773 of our 2,088 Model Descriptions are a MathWorks copyright notice.

For all comments of Table 3, we identified the duplication origins, *i.e.*, why the comment's text appears more than once. Based on our manual analysis, we hypothesized five types of duplication origins (based on [5] and our observations):

generic: a comment's text is very short or non-specific, making it likely that it appears more than once, *e.g.*, all Element Descriptions listed in Table 3,

copy-paste: the comment or the comment text was copy-pasted within the model or from model to model, *e.g.*, the most copied DocBlock of Table 3,

library: the comment is part of a library (only possible for Element Descriptions, DocBlocks, Annotations), *e.g.*, all Annotations of Table 3,

IDE generated: the comment or comment's text was generated via the IDE (*i.e.*, the IDE starts stubs for the user to fill in, or gives generic info), *e.g.*, "UNTITLED Summary of this class goes here \nDetailed explanation goes here,"

synthetically generated: we found a number of comments in MATLAB code that were synthetically generated. In fact, in all instances of synthetically generated comments we observed, the complete code files were synthesized, *e.g.*, "rad" and "Translation Method - Cartesian."

We often could not confidently categorize whether a comment was copy-pasted or just generic as we only observe the final identical texts and not the duplication process, and thus conservatively united the categories in Table 2. Only a few of the heavily duplicated comments are generated by the IDE or synthetically. Overall, one can see a divergence in the categories *generic/copy-paste*, *library*, and *synthetically generated* for the different types. The last column of Table 2 shows that taking only the top ten most duplicated comments, *e.g.*, Element Descriptions, represents already a high percentage of all comments of its type. This fact gives another perspective to interpret Figure 3.

Table 2: Overview of the ten most duplicated comments' duplication reason per comment type. The last column shows the ratio of top ten duplicates and the total number of comments of that type.

Comment Type	Generic/ copy-paste	Library	IDE	synthetic	$\frac{\text{top10}}{\text{all}}$
Model descr.	10	0	0	0	0.42
Element descr.	4	6	0	0	0.59
Docblock	7	3	0	0	0.53
Annotation	0	10	0	0	0.18
Class comment	9	0	1	0	0.17
Other comment	4	0	0	6	0.05
total	34	19	1	6	0.11

Comments at Different Levels of the Subsystem Hierarchy

We give a breakdown of the commenting practices at different levels of depth of the subsystem hierarchy in Table 4. We define Model Descriptions to occur at the hypothetical depth 0 to include them in the table. One can see that most Element Descriptions are located at depth 4 and Annotations peak at level 3. DocBlocks are the only comment type with two local maxima at level 1 and level 7, respectively. In absolute terms, most comments occur at depth 3, while deduplicated, most comments lie at the root level (depth 1).

While the ratio of comments per subsystem is highest at the root level, the Model Descriptions at depth 0 lead to the highest ratio of comments per element. The ratio of comments per subsystems drops from depths 1 to 4, stabilizing afterward.

Comparing the columns of *comments* and |*comments*| shows that the highest ratio of original comments can be found in the upper levels, with 75% of Model Descriptions and less than 50% of the comments at the root level being duplicates. At the other extreme are depths 10 or more, with $> 98\%$ duplicated comments, which is why we cut them from Table 4.

To not skew our analysis of comment lengths, we used only the deduplicated comments to compute the mean and median lengths in the last two columns. At all depth levels, the mean length (denoted by $\bar{x}_{len}$ in Table 4) of a comment is longer than its median (denoted by M_{len}), indicating a positive-skew (right-tailed distribution) of comment lengths. The mean length of Model Descriptions (depth 0) is much longer than any other comment. Similarly, the root level's mean comment length is about twice as long as on deeper levels of the subsystem hierarchy. Median lengths do not show a clear trend, with only the Model Description, again, being much longer than the rest.

As Annotations can both be containing text of various lengths, but can also be highlighting areas without text, we analyzed how Annotations are primarily used. We found that very few (0.2%) of the Annotations are highlighting an area only, *i.e.*, not holding a single comment text character. If used, such area-only Annotations are often highlighting a group of blocks (and not only empty model canvas). Overall, there are also few (8.1%) Annotations, containing one or multiple blocks, showing that most often Annotations are used as a purely textual companion, next to other model elements. Those Annotations that contained blocks usually hold a comment that is 10 to 100 characters long.

Comment Guidelines

We investigated, whether developers followed the guidelines we gathered in Section 2.4. We skipped those guidelines that are not objectively measurable: guidelines concerning comment appearance and formatting, subjective guidelines about "meaningfulness" of comments; or guidelines regarding generated code, unobservable for us.

`himl_0001` (standard header comment) No source code file of our data set features the standard header of G1.

`himl_0003` (comment density: 0.2 comment lines per line of code) We observed a higher mean of 0.271 comment lines per line of code, and a median of 0.25.

`jc_0603` (model description) We found only 2,088 of 9,033 models having a Model Description, while many of them are generic or copy-pasted duplicates, see Tables 1 and 2.

Table 3: The most duplicated comments in our data set listed by type. We marked shortened comments by [...], and new lines by \n.

Comment Type	number of occurrences	Comment Text
Model Description	247	Copyright 2017-2018 The MathWorks, Inc.
	117	Copyright 2014-2018 The MathWorks, Inc.
	95	Thomas Modules
	77	Copyright 2015-2018 The MathWorks, Inc.
	75	Copyright 2014-2017 The MathWorks, Inc.
	73	\nCopyright 2014-2016 The MathWorks, Inc.
	68	\nCopyright 2009-2018 The MathWorks, Inc. MathWorks, Inc.
	45	\nCopyright 2015-2017 The MathWorks, Inc.
	42	\nCopyright 2014 The MathWorks, Inc.
	39	\nCopyright 2013 The MathWorks, Inc.
Element Description	748	Initialise
	436	Output Signal
	342	Input Signal
	321	\nStore in Global RAM
	259	Add in CPU
	222	source block
	200	Trigger
	197	Fader Output
	196	Lower Limit
	196	Upper Limit
DocBlock	51	This subsystem computes the surge, sway, heave, roll, pitch and yaw motions of the center of the body [...]
	51	This subsystem computes the elevation of the sea wave, where the sea wave spectrum is given by [...]
	17	These are the Wave Excitation Forces computed by WAMIT-Demo version \nthe angle is between [...]
	11	jza - 21.08.07 The test model is created manually. \n\nTransformation rules for test data variants [...]
	8	Integral de sinal seno = sinal -coseno
	7	**Steps to Create a Quartus VHDL project****Simulink Steps**1. Setup all the paths
	6	Some text about the spec... \n
	5	Derivation of State Space model from original equations
	4	By testing SyD, you will be able to discover its advanced features and advantages
	2	Synchronous machine\r\n>>> Power conserving transformation
Annotation	3,840	The Measurement is not modified
	1,837	Pierre Giroux, Gilbert Sybille\nPower System Simulation Laboratory\nIREQ, Hydro-Quebc
	1,725	1) Only subsystems can be added as variant choices at this level\n2) Blocks cannot be connected at this [...]
	1,539	=
	1,464	Graphical user interface for the analysis of\nSimscape Power Systems \nPlace the Powergui block in the [...]
	1,434	*
	1,196	U(k)
	1,090	[d\n q]
	1,090	[al\n be]
	954	Integrator
Class Comment	15	Author: Colin Eles elesc@mcmaster.ca \n Organization: McMaster Centre for Software [...]
	12	Copyright 2014 The MathWorks, Inc.
	8	Author: Matthew Dawson matthew@mjdsystems.ca\n Organization: McMaster Centre for [...]
	8	Copyright (c) 2016, The MathWorks, Inc.
	8	%%% [...]
	7	Copyright 2014 - 2016 The MathWorks, Inc.
	7	CONNECTIVITYCONFIG PIL connectivity configuration class\n\n Copyright 2018 Arm Holdings
	6	%%% [...]
	6	UNTITLED Summary of this class goes here \nDetailed explanation goes here
	5	Copyright 2017 The MathWorks, Inc.
non-Class Comment	1,524	\n
	1,368	rad
	1,359	Translation Method - Cartesian\nRotation Method - Arbitrary Axis
	1,130	in
	954	m
	594	Las unidades de la resistencia son "Ohmios".
	531	User supplies all inputs
	431	kg*m^2
	398	Inertia Type - Custom\nVisual Properties - Simple
	398	%

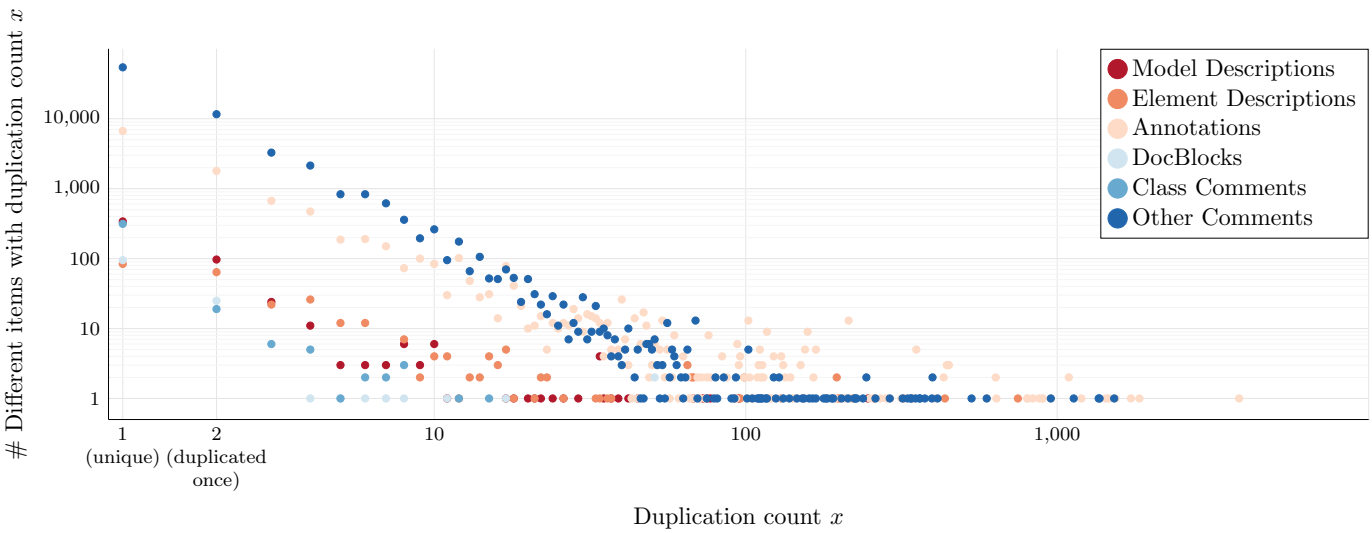


Figure 3: Scatter plot of duplication counts and the number of their occurrences. Note: both the x–axis and y-axis are logarithmic.

Table 4
Occurrences of Simulink comments (except Notes) at different subsystem depths.
[1] Mean and median lengths in chars of the row-wise deduplicated $|comments|$. [2] Each model is counted as one subsystem and element at depth 0 for Model Descriptions. [3] The only subsystem at depth 1 is the root subsystem.

depth	Model Descriptions	Element Descriptions	Annotations	DocBlocks	*comments*	$\frac{comments}{subsystem}$	$\frac{comments}{elements}$	$\lvert comments \rvert$	$\tilde{x}^1_{len}$	M^1_{len}
0	2,088	0	0	0	2,088	0.23[2]	0.231[2]	521	174.91	71
1	0	622	10,965	132	11,719	1.30[3]	0.016	6,202	98.98	32.0
2	0	875	13,425	24	14,324	0.32	0.007	3,162	57.41	17.0
3	0	1,333	16,260	25	17,618	0.24	0.007	2,091	54.30	22
4	0	1,925	12,750	8	14,683	0.17	0.005	1,257	52.55	22
5	0	431	13,404	0	13,835	0.18	0.006	869	55.49	23
6	0	90	10,436	17	10,543	0.21	0.006	536	58.41	24.0
7	0	20	6,909	68	6,997	0.22	0.006	263	60.48	30
8	0	6	3,220	34	3,260	0.10	0.003	81	71.01	41
9	0	1	2,317	0	2,318	0.17	0.007	58	48.40	38.0
total	2,088	5,303	91,027	308	98,726	0.23	0.007	12,255	85.84	30

***RQ 1**: How are Simulink projects documented?*

Annotations are the most used Simulink comment feature, while Notes are barely used. In MATLAB, there are few class comments. All types of comment show high numbers of duplicates, but each comment type (except Element Descriptions) has more unique comments than comments with at least one duplicate. Simulink models have the highest comment density at model and root level of the subsystem hierarchy for all comment types; the longest, least duplicated comments are also there. At lower depths, comments are often duplicated, but the density or comment length does not drop off. Few comment guidelines exist for Simulink or MATLAB; most not objectively measurable. MATLAB is commented more than guidelines demand, few models come with a model description, and the standard header is not featured in MATLAB code.

***RQ 2**: Does the amount of documentation vary in different models?*

Here, we compute the correlation matrix of model size, cyclomatic complexity [43], and age as well as amount of model comments. We break down model size into overall number of model elements (blocks, signal lines) and number of subsystems, and use a model's age as a proxy for its time under development. We also break down 'the amount of comments' into number of comments, the total comment length in chars of a model, and the mean and median comment lengths of a model.

The correlation matrix of these metrics is given in Figure 4. As none of our metrics are normally distributed, we employ Spearman's rank correlation coefficient. We only consider higher correlations between two metrics, and ignore weak correlations $\rho < 0.3$ or too low significance levels of $p < 0.05$ (note: $p \neq \rho$).

Most correlations are significant: strong correlations are shown in color, weak correlations in gray. A few correlations are insignificant, shown in white. Only a single negative, albeit somewhat weak, correlation is present between the number of comments and the median comments' length of a model. There are only two comment metrics showing correlations to the model size, complexity, or age metrics: number of comments and total number of comment chars of a model. Lastly, time under development is uncorrelated to any other metric we measured.

Spearman's correlation only measures correlations of ranks and not of actual values. This is why we also give an overview of the distributions of the metrics of maturity and comment elaborateness, which show a strong positive correlation in Figure 5. This shows, whether the values also grow somewhat similarly. For each of the metrics, we give the mean value of each quintile of their distribution. For example, if one sorts the models by the number of elements (the left-most five bars), the quintile of smallest models only has 11 elements in the mean, while the second quintile's models are bigger with 38 elements, *etc.* One can see that all metrics from our selection are strongly positive-skewed, as they grow from quintile to quintile even with our logarithmic y-axis. The last quintile features a "growth spurt" for all metrics. This "growth spurt" is especially drastic for the number of elements, subsystems, and comments. While none of the metrics is a complete outlier in terms of growth, one can see that the complexity does not grow as fast as the other metrics. Similarly, one can see that the total comment length does not keep up with the growth in the upper quintiles. Finally, this chart shows that most models only have a handful of comments, overall.

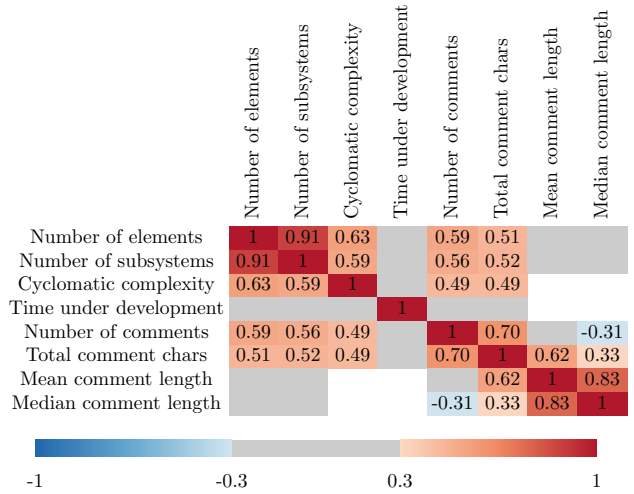


Figure 4: Heatmap of rank correlations of maturity metrics and comment amount metrics. Weak correlations with $|\rho| < 0.3$ are depicted in gray, and insignificant correlations with $p < 0.05$ are shown in white.

***RQ 2**: Does the amount of documentation vary in different models?*

The number of total comments and total comment length of a model grows as the model grows in size (number of elements/subsystems) and complexity. Other correlations are either weak or insignificant. In particular, time under development does not correlate to any other metric we measured.

***RQ 3**: How can the content of Simulink comments be classified?*

Deriving SCoT from CCTM

While working on RQ 3, we started by adapting the CCTM's terms slightly to fit our context. This means that we changed terms like "source code" to "model" for Simulink, and adjusted terms of the CCTM only referring to "classes". We also decided on clear boundaries to differentiate between the categories *Summary* and *Expand*.

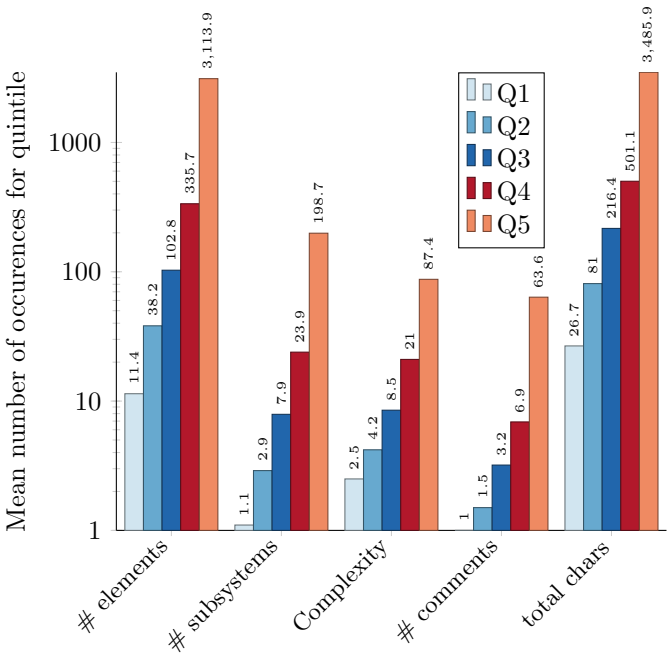


Figure 5: Mean quintile values of metrics that showed correlation.

In the CCTM, a *Summary* is a brief description of functionality and purpose, covering the question word 'what'. The *Expand* category is used to provide more details on the code to answer the question word 'how'. In practice, we found it hard to differentiate these categories and thus decided on a more objective criterion. To this end, we defined the category *Summary* to be a title of a module or summarizing at least 10 model elements or source code lines, while being at most 3 sentences long. We used the *Expand* category for the remaining candidates that were longer or described fewer elements/lines.

A few of our classified comments in Simulink and MATLAB did not fit in any prior CCTM category. To classify these comments accurately, we introduced five new categories (shown in italics in Table 5):

IDE Hint: (higher-level category notice) an instruction of how (not) to use the IDE to achieve certain results.

System Requirements: (notice) description/list of hardware or software requirements that make it possible to use the artifact and all its features.

Version History: (metadata) a description of older versions, version names, and dates of changes. This category is partly covered in the *Deprecation* category in the CCTM.

Interactive: (media) a comment which helps developers to interact with the program or IDE, such as interactive buttons in a comment that start or stop the simulation of a Simulink model.

Picture: (media) a picture, illustration, or figure for documentation purposes, such as a screenshot or example output.

Note that we created a new higher-level category *Media*, which is easily extendable for different kinds of media; other languages may use for documentation, *e.g.*, audio, video, *etc.*

While classifying, we came upon calls to action like "in case of bugs, please contact us at `adress@mail.host`". We expanded the *Ownership* category to cover such contact requests instead of creating a new category.

In the classification process, we decided to discard non-English text, as we could not ensure our complete understanding in categorizing such comments. We found text in Japanese, German, Dutch, and Spanish showing the diversity of the Simulink and MATLAB communities. As can be seen in Figure 6, the *Discarded* category was one of the smallest for both languages.

Simulink and MATLAB Comment Information

The detailed results of our manual classification of Simulink and MATLAB comments are listed in Table 5. It can be seen that the lower-level categories *Summary* and *Expand* (from *Purpose*) are most often utilized, with the categories *Usage* and *Ownership* still being used for more than a tenth of comments. Overall, 22 categories are covered by our samples (19 by Simulink, 16 by MATLAB). We do not show the CCTM categories *Deprecation*, *Incomplete* and *Directive* in Table 5, as we found no instances of them in any of our samples.

From the 374 Simulink comments we analyzed, 59 covered more than one category. Many of such multi-topic comments were visually split into different parts by line breaks, where one part covered, *e.g.*, a *License Information*, followed by a *Summary*. Overall, we used 458 category classifications for our 374 Simulink comments (1.22 categories per comment). In MATLAB's 383 comments, on the other hand, 108 comments covered more than one category, totaling 630 categories (1.64 categories per comment).

An aggregation into higher-level categories of Table 5 is shown in Figure 6. While *Purpose* dominates across both Simulink and MATLAB, *Notice*, and *Style/IDE* still cover more than every seventh comment in each language.

Table 5: Detailed overview of the manual classification of our sample set: 374 Simulink and 383 MATLAB comments. The columns add up to more than 374 or 383, because a single comment can cover multiple categories. New categories of our taxonomy are printed in italics and unused categories of the CCTM are not shown.

Higher-level Category	Category	Simulink	MATLAB
Purpose	Summary	118	108
	Expand	131	235
	Rationale	11	17
Notice	Usage	93	56
	Exception	2	0
	IDE hint	2	0
	System requirements	0	5
Under Development	Development notes	11	17
	Todo	0	2
	Commented code	1	35
	Coding guidelines	1	0
	Extension	1	0
	Recommendation	1	2
Style & IDE	Formatter	6	30
Metadata	License	0	9
	Ownership	35	49
	Version history	6	19
	Pointer	16	25
Discarded	Auto generated	1	2
	Noise	15	19
Media	*Interactive*	6	0
	Picture	1	0

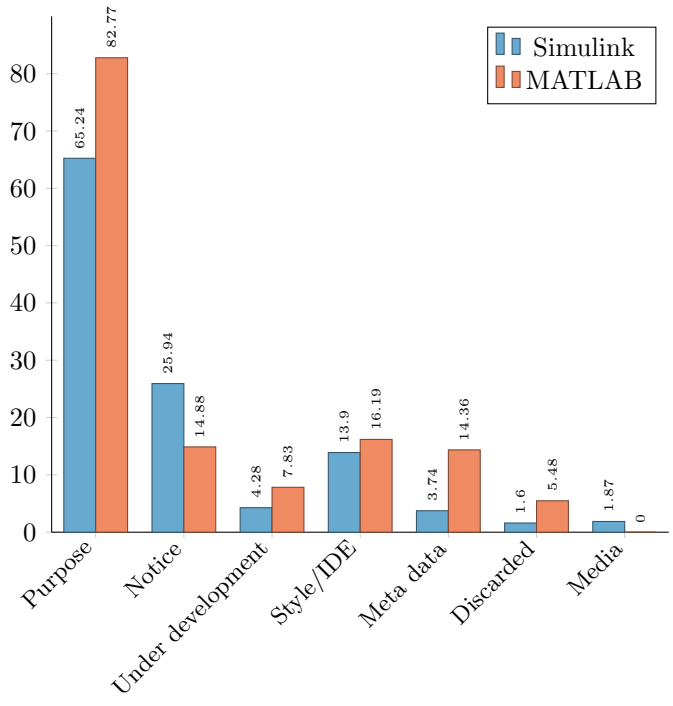


Figure 6: Higher-level category distributions of our sampled comments of Simulink and MATLAB. Note that the percentages sum up to more than 100%, because a single comment can cover multiple categories.

***RQ 3**: How can the content of Simulink comments be classified?*

The CCTM taxonomy is mostly applicable to Simulink and MATLAB. We added the categories *IDE Hint, System Requirement, Version History, Interactive*, and *Picture*, while the categories *Incomplete Comment, Directive*, and *Deprecation* were not applicable. This yields our taxonomy SCoT. Simulink and MATLAB both cover nearly the full breadth of the CCTM taxonomy. Comments from the *Summary*, *Expand*, *Usage*, and *Ownership* categories dominate in both languages. Simulink comments are more narrowly focused per comment, as, on average, each comment cover only 1.2 categories, while a MATLAB comment covers 1.6 categories.

***RQ 4**: How does Simulink documentation compare to textual programming languages?*

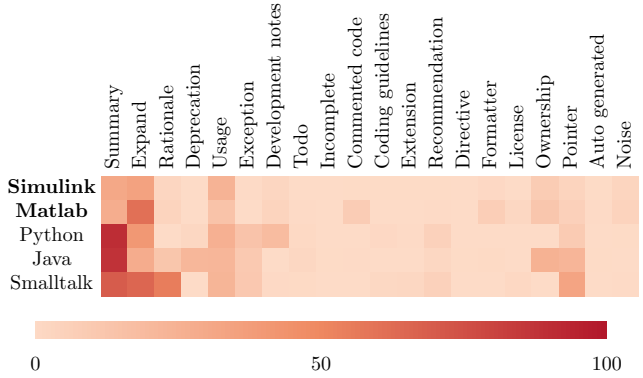


Figure 7: Heatmap comparing our CCTM categorization (highlighted in bold) and previously categorized languages. The categories of the CCTM are listed horizontally, while the color scheme depicts how many percent of comments fall into a category.

This question is partly answered by our answer to RQ 3: we employed the CCTM taxonomy, with only slightly adjusting descriptions of the categories and adding five seldom-used categories to the pre-existing category set. This shows that comments in textual and visual languages mostly cover the same categories.

A comparison over the distributions of classifications is shown in Figure 7. To make a comparison between the different languages possible, we use the category mapping found in Fig. 8 of Rani *et al.* [39]. The top rows of Figure 7 are very similar to a heatmap version of Table 5. The difference is that we use another category set in Figure 7: (1) we do not show our new categories as these were not part of the CCTM and could not have been found in Python, Java, or Smalltalk by definition, (2) we include categories that were used for Python, Java, or Smalltalk, which we did not find in our samples from MATLAB or Simulink. Overall, one can see a similar distribution between the

languages, *e.g.*, the categories *Summary*, *Expand*, and *Usage* are heavily used in all languages. From the languages studied in this work, we found that they lack in *Exception* comments, compared to the other languages. MATLAB features more *Commented Code* and *Formatter*, than all other languages.

In RQ 3, we reported that, on average, a Simulink comment covers 1.21 categories, while MATLAB comments cover 1.64 categories. These results compare to the previously studied languages as follows: Python 2.23, Java 2.47, and Smalltalk 2.91 (derived from data of Rani *et al.* [39]).

> ***RQ 4**: How does Simulink documentation compare to textual programming languages?*
>
> Simulink and MATLAB comments cover mostly the same breadth of categories as Python, Java, and Smalltalk comments. In addition, each lower-level category was chosen similarly often for each language.

5. Discussion

In this section, we discuss our main findings, new insights, and possible implications, structured by research question.

***RQ 1**: How are Simulink projects documented?*

General Measurement and Properties

We found that Annotations are the most common comment type in Simulink by far. This could be due to Annotations being the only type of comment showing the content directly in the model window. While adding a new Annotation, developers do not have to switch to another window and can use Annotations for several purposes (*c.f.* Section 2.2) directly in the Simulink IDE. Readers of Annotations are also directly aware of the presence of Annotations and are able to read their content without opening a new window, as they would have to do with the other comment types. This impediment may explain the relative lack of instances of DocBlocks, Element Descriptions, and Notes. An additional reason for Notes is that they are the newest commenting feature in Simulink, only present since 2018, with the SLNet dataset [46] being gathered in 2020.

With 1,773 of 2,088 Model Descriptions featuring a MathWorks copyright notice in our data set of 9,033 models, we find the Model Description feature mostly unused, outside of MathWorks models.

In our view, some comment types in Simulink show usability shortcomings: comment types whose presence is not indicated to users immediately (Model Description, Element Description, Notes), or their content not directly accessible (DocBlocks) are hard to handle, or it is cumbersome to discover their existence. For example, users have to perform two clicks to see whether an element has a description, or not. We doubt that users would try to find out one by one which elements of a model contain an Element Description. A Model Description requires four mouse clicks to access, but there is only one Model Description per model, giving it a central place. DocBlocks are shown in the model, and users will thus see that some form of comment is present, but the content is opaque until accessed by a double click and waiting for, *e.g.*, Microsoft Word to open. Users may also need to install an `.rtf`-editor to access the content.

In view of all this, we suggest improving the accessibility of Element Descriptions by adding a small symbol on documented elements, or on a mouse-over to highlight the element or display the comment text. This ensures that developers become aware of an Element Description. DocBlocks similarly could display their (unformatted) comment on a mouse over, without opening an external editor window.

An alternative approach could be to refrain from using any other type of commenting feature apart from Annotations and Model Descriptions. This makes comments directly accessible in the case of Annotations, and gives a central documentation location to find and automatically process vital model metadata in the case of Model Descriptions. Limiting the set of comment types could also help developers in their choice of which of the five comment types to use to document a particular aspect of their model.

While conducting this study, we asked Mathworks developers whether they view any of the commenting features as obsolete or to be preferred, in private communication. A Mathworks engineer disclosed to us that Mathworks views none of the comment types as obsolete, per se. The Mathworks engineer added that DocBlocks can viably be replaced by a Model Description, Annotation, or a Note, though.

We found class comments in 86% of MATLAB classes. This stands in contrast with previous findings [39]: 68% in Java, 23% in Python, and 38% in Smalltalk. As the class feature is seldom used in MATLAB (only 552 classes in 17,792 MATLAB source code files), it is an atypical phenomenon. This could explain the outlying percentage level. Note that we have checked that the MATLAB IDE does not create class comments automatically.

Comment Duplication and Duplication Reasons

Many comments in our study set are heavily duplicated, but different comment types are duplicated in different ways. For instance, we only found MATLAB comments to be synthetically or IDE generated. Some duplication actually is unavoidable, even with good commenting practice: Element Descriptions refer only to a single, often simple model element and, therefore, are expected to be more simple and similar to each other than more complex structures such as comments for subsystems or complete models. Following up on this, the high amount of generic/copy-pasted Model Descriptions seems to be haziness by developers: we

found many simple copyright statements without any information concerning the specific model itself. We suggest that developers should follow guideline `jc_0603` (see Section 2.4), and be even more specific about the information of the model description: give at least a title and short purpose description of the model in addition to author and copyright information to each model.

Comments at Different Levels of the Subsystem Hierarchy

We found that the most elaborate, least duplicated comments occur at the root level of models. This is also the place with the highest comment density. Comment length does not change from the second layer downwards – only the frequency of comments at depths two and three is slightly higher than at lower depths. All this suggests that developers put more effort into documenting the top level(s). This might be because the root level and Model Description offer the possibility to document the complete model at once at a central place, which is easy to find. In contrast to this, lower-level comments may focus only on the direct context, *i.e.*, not the surrounding subsystems of higher or lower levels, and thus are shorter. Similarly to our findings, in Java, higher level comments (class, file, interface) have a higher density than method comments [48], while method comments are longer and show a higher comment density than the lower level inline comments [17].

Comment Guidelines

Our search for guidelines on documenting in Simulink and MATLAB returned only sparse results. In particular, novice developers would not be guided in most documenting decisions, *e.g.*, which elements to comment, what comment type to choose, or where to document. Of the three guidelines of which we tested developer adherence, only one was followed. We suspect that the official guidelines are not well known, or mostly ignored, by open-source developers. This suggests that developers employ their comments ad-hoc and comments differ from project to project.

We recommend giving clear advice on when to use which of the many Simulink comment options. We suggest investigating in more depth why developers currently mostly use Annotations and hardly use any of the other Simulink commenting features. We also suggest having one designated Annotation per subsystem for the subsystem *Purpose* (*Summary* and *Extend*) in a designated corner, *e.g.*, top left. This way, developers would know where to look for the most frequent information. One way to help developers and nudge them into employing such Annotations would be to automatically create this designated Annotation, partially pre-filled, at the very moment a new subsystem is created.

Our last guideline suggestion is to always attach an Annotation to a model element or a group of elements. If Annotations are tethered to another element, they cannot get lost or be forgotten about as easily if a model is refactored or otherwise modified (*i.e.*, documented elements are moved, copied, or deleted). Developers reading a diagram do not have the added burden of inferring which comment is referring to which element. Also, once an element and its comment have no connection (anymore), reattaching them presents challenges [44]. The title and purpose Annotations we proposed should then be tethered to a whole subsystem. To not overwhelm users with documentation text, we suggest to give annotations the new feature of minimization. This way, developers can elaborate design particularities or anything else at length, without cluttering the view canvas.

***RQ 2**: Does the amount of documentation vary in different models?*

Answering RQ 2 gave us interesting insights into the (non-)correlation of various model metrics. For example, there is no correlation of model size or complexity to the length of comments. As models evolve, more model elements get added, than removed [45]. Only the number of comments and the total length of comments increase with models becoming bigger. Taking this together, it means that as a model grows, developers do not add to existing comments but add new ones instead. Based on our findings, it is unclear whether the existing comments get further updated, as their length remains the same. However, Jaskolka *et al.* found that Simulink comments are among the least changed elements of Simulink models in their industrial study [20]. This also mirrors findings in textual programming languages, where comments are not updated along with their corresponding code [54]. This fact sets comments up to be out of sync with its corresponding code or model.

Furthermore, we observed that a model's age is not correlated to any other metric of our study. This indicates that open-source projects either develop their models (not only comments) at very different speeds or do not consistently work on their models.

We saw a negative correlation between the number of comments and their median length (also a negative, albeit weak correlation to the mean length). This indicates that developers compensate for creating a higher number of comments by slightly shortening each. In our manual classification, we sometimes found short Annotations visually grouped tightly together, forming a connected documentation text if one unites the related Annotation texts of the group. Some developers used these individual Annotations to format a text, because each Annotation can be moved freely on the model canvas, so that it aligns to the developer's wishes.

We can see in Figure 5 that, in the upper quintiles, the number of comments and total comment length do not grow faster than the models themselves. This means that there is no relative increase in commenting effort in the biggest models. Before conducting this study, we had the hypothesis that bigger models are built by more professional teams, which would put more effort into comment-

ing. This does not seem to be the case, at least in open-source models.

Other observations, like the correlations of size metrics and cyclomatic complexity, align with correlations of lines of code to cyclomatic complexity found in Java, C, and C++ [13].

***RQ 3**: How can the content of Simulink comments be classified?*

When answering RQ 3, we found that the CCTM taxonomy covers a wide breadth of comments of Simulink and MATLAB, already. We also found it to be easily extendable. In our samples, only five seldom-used categories needed to be added to form SCoT. As our work did not focus on just class comments, but, in contrast to prior work, also considers models that are often designed by non-software engineers, we view this to be only minor additions. We thus expect the SCoT to be applicable in projects using other languages with only minor adjustments.

The most frequently used higher-level category from the SCoT for both Simulink and MATLAB is *Purpose*, showing that developers mostly care about documenting "what is the code about".

In our manual classification process, we found (and discarded) very few non-English comments. Even though, we did not classify them, we briefly analyzed them after an ad-hoc translation and found them to have similar information and format to English comments. We therefore believe that such non-English comments could be classified similarly to English comments.

Regarding our additional categories for extending the CCTM in RQ 3, we note that we only found few instances of the *Version history* category. We expect more sophisticated projects (as were studied prior with the CCTM) to usually handle the aspects of *Version history* either in release notes or directly in the VCS' commit history. We expect the new category *IDE Hint* to be used only for languages that use a common IDE. Both MATLAB code and Simulink models are commonly used in the MATLAB ecosystem, as a working and licensed MATLAB installation is necessary for their execution, anyway.[9] Lastly, the new higher-level category *Media* holds the Simulink-specific sub-categories *Interactive* and *Picture*. Classical programming languages are limited to text-based comments. However, previous work [35] has found media, such as images, in README files, which shows developer interest in expressing their documentation in different forms. In general, we expect other languages to enable commenting via media like audio or video in the future. Our new higher-level category *Media* can be extended with such modalities, easily.

[9]There is a plugin for MATLAB in Visual Studio Code, but only very basic features of code editing are available without a working MATLAB installation.

We imagine that comments of the *Interactive* category can be extremely useful in program understanding – both of abstract purpose and inner design. Various modes or parameters of the model can be preset, and their execution can be discovered immersively. Such interactive documentation thus offers the possibility to "show, not tell".

While answering RQ 3, we manually classified each item. Prior work [44] already derived heuristics to identify some categories like Summary, Ownership, and Expand, for the diagram language Ptolemy,[10] with some success. We expect that similar heuristics could be divised for most of our categories. Similarly, we expect LLMs to be applicable for automatic classification, see also Section 7.

***RQ 4**: How does Simulink documentation compare to textual programming languages?*

The findings from RQs 3 and 4 demonstrated substantial similarities in both quantitative and qualitative terms between Simulink and MATLAB commenting as well as Python, Java, and Smalltalk. This shows that Simulink, although a visual language with a diverse comment feature set, is, in fact, documented similarly to textual languages.

While comparing Simulink and MATLAB to Python, Java, and Smalltalk, recall that the prior studies focused on class comments from high-profile projects. This showed most prominently in that class comments covered more of the CCTM categories per comment than in our sample. This seems intuitive, as class comments are longer and more exhaustive than other code comments. In fact, we expect comparing class comments in Python, Java, and Smalltalk with MATLAB class comments and Simulink root subsystems' DocBlocks, main Annotations, and Model Descriptions to yield similar results.

Quantitatively, the different languages showed a very similar distribution *c.f.* Figure 7 – even though different research teams studied different languages, different comment types, and different project types. For us, this is an indication that commenting cultures are similar even while crossing so many boundaries. We thus expect that there is significant potential for knowledge transfer between findings from comments in textual languages to visual languages, and vice versa.

6. Threats to validity

6.1. Internal Validity

Although our manual classification process for RQ 3 is subjective, we mitigate this threat by conducting a triple-review process with a majority vote and group discussions for unclear comments, similar to prior work [39]. By employing this technique, we strive for a more objective classification. A summary of our classification process is shown

[10]https://ptolemy.berkeley.edu/ptolemyII/index.htm

in Table 6. Around 20% (150/757) of the reviews objected that a comment's category was missing, too much, or wrongly classified. In the second step, the original evaluators judged the reviews themselves and accepted about 75% of them. This left only 38 comments, where a third reviewer made a final decision after weighing both the evaluation and review. While the evaluation and review phase was evenly distributed by design, the steps afterward depended on the decisions of these two phases. For example, E3 had the highest agreement rate for reviewing MATLAB comments, *i.e.*, they issued only few objecting reviews to the original evaluation.

Some Simulink comments or MATLAB comments are part of a larger context of related comments. These are usually graphically close, or in a code line nearby. Our scripts to collect and sample comments could not link such "related" comments, and they were thus gathered in isolation. However, in our manual classification, we inspected each comment, and could thus see, whether a nearby comment was part of the context of our comment to classify.

By answering RQ 2, we found a model's time under development not correlating to any other metric, we computed. We hypothesize that Simulink may compute this time faultily in some cases. On inspection of the times, we only found 56 times from our 9,033 models to be obviously erroneous, though. These either had a negative time under development or one of less than ten seconds – we excluded them prior to our analysis in Figure 4. All correlations of time under development that are too weak (shown in Figure 4 in gray color), are positive. This indicates that the metric can be assumed to be correct, overall.

Table 6: Overview of the classification process. While 757 comments underwent an evaluation and review, only 150 of the reviews elicited objections, and of those only 38 were not accepted by the original evaluator and thus needed a final decision.

	Evaluator	evaluated comments	objecting reviews	final decisions
Simulink	E1	124	25	5
	E2	124	28	6
	E3	126	21	12
Matlab	E1	127	33	5
	E2	128	35	1
	E3	128	8	9
total		757	150	38

6.2. External Validity

Our analysis set consists of open-source projects from GitHub and Mathworks Central. Comments in industry-projects may differ significantly. Via industrial acquaintances, we know that some companies have internal guidelines but do not know whether these cover comments and how they would employ comments in Simulink. Still, our data set is highly diverse, comprising everything from toy projects to industry-like projects [7], and thus gives valuable insights into how Simulink comments are used in practice.

7. Related Work

7.1. Comment Analysis

Code comments are an active research topic which has evolved over decades. Already in 1976, Boehm *et al.* [6] started to develop metrics predicting software quality from quantitatively measuring source code commentary. In particular, they doubted that comment length alone is an indicator of good software. They also already gave advice of not over-explaining some code at the expense of leaving other code uncommented. Lastly, they describe a smell detecting tool "CODE AUDITOR", which checks source code for coding standards, *e.g.*, missing header block comments. In 1978 Krogh [25] not only demanded the presence of code comments, but also certain qualities of code comments: in the terminology of our paper, Krogh demanded comments of software *Purpose* and *Usage*, and also gave some examples of *Pointers*.

Since then, the research community studied a multitude of aspects of code comments. Some aspects are: the importance of code comments for readability, extensibility [9, 32], comment coherence [49], comment consistency [54], comment completeness [18], and comment adherence to coding guidelines [36, 52]. In the last decade, research on code comments often focuses on assessing the comment quality itself [22, 49], classifying comments automatically [34, 44], completing them [59], updating them [27, 28, 44], or even generating them [16, 19]. Such approaches often employ machine learning techniques, which mine code and comments from open source software projects, to create a learning database.

Our work employs a taxonomy for classifying class comments from Rani *et al.*, called Class Comment Type Model (CCTM). They employed their taxonomy on Smalltalk classes [40], but also gave a mapping of their taxonomy [39] to prior taxonomies used for Java and Python [34, 58]. In our work, we slightly adapt and extend the CCTM for our study set of Simulink and MATLAB projects. Kostić *et al.* give an overview of code comment taxonomies [24] and used a proposed taxonomy to classify multi-language comments [23]. However, their taxonomy is much more coarse-grained, than the CCTM.

Blasi *et al.* [5] studied comment duplication (Type I, III comment clones) in Java source code. They strived to identify problematic clones that were too generic or copy-pasted. We only searched for Type I comment clones in Simulink and MATLAB. Our classification also did not aim at finding problematic duplications, but at finding the duplication origin. We thus classified comment duplication as generic/copy-paste, library imports, IDE generation, or synthetic generation.

7.2. Simulink Comments

There has also been some prior interest in studying comments in Simulink. Pantelic *et al.* studied industrial Simulink projects and their evolution, as well as commenting practices [20, 33]. In [20], they studied the frequency of changes on various model comments during the model's development. They found that comments were least often changed. Within the comment changes, Annotations were changed most often, while DocBlocks remained mostly static. Pantelic *et al.* did not study the frequency of changes on Element Descriptions (block description, signal description) or Notes – we considered both features in our study. They also did not analyze the actual comment information or other characteristics like lengths or duplication. As they studied an industrial project, the experimental data and most basic information about the project itself is not available. In their anecdote-driven work [33], Pantelic *et al.* argue that current Simulink modeling practice faces several challenges: a lack of automation, (high quality) tools, and documentation templates. In fact, even a standard process of documentation is missing in a culture of prototype first, documentation third (or never). Pantelic *et al.* refute that (Simulink) models *are* already documentation, as the model only provides syntactical understanding, while documentation provides additional semantic understanding. They demand good documentation providing information about (1.) *software requirements specification*, which should give a model's black box behavior in a more abstract way than the direct implementation; and (2.) *software design description* (SDD), which should give semantics about the internal design, anticipated changes, hierarchy, and interfaces. The research group around Pantelic also developed a template for including SDD information into the model and a tool helping with the documentation process [42]. DocBlocks are created automatically, so that the developers can manually enter the documentation into a designated location. Their tool creates such DocBlocks for Purpose, Internal Design (focusing on interfaces), Rationale, and Anticipated Changes (see SDD, above). Developers are also expected to document changelogs and system acronyms/notation/definitions. Overall, their template covers the most-used categories of the CCTM used in our work.

While there are some studies, collecting open-source Simulink models [10, 46], and providing various metrics of models [3, 7, 47], none of those studies analyzed Simulink comments.

To the best of our knowledge, we are the first to study the commenting practice in open-source Simulink projects, as well as analyzing actual comment information of Simulink models. We are not aware of studies of comments in other visual modeling languages like UML or SysML.

8. Conclusion and Future Work

In this study, we found that open source MATLAB and Simulink projects feature a wide variety of types of comments, covering nearly the whole spectrum of the commentary taxonomy CCTM in addition to others. Many of the comments are duplicated by various means and are present in all levels of the model hierarchy, while developers focus mostly on the highest levels. We have shown that bigger and more complex models feature more comments and a higher total comment length, while each comment does not change in size. Model age on the other hand is neither a factor in model size nor comment amount. Finally, we found that the CCTM taxonomy is applicable for languages of different paradigms, and we extended it into a more complete taxonomy, named SCoT. We expect SCoT to be useful for classifying comments of all types, and languages, while probably needing only slight adjustments or additions.

We found comments in Simulink to only stand out in their many comment types in comparison to textual languages. In terms of information diversity and distribution, Simulink comments fall in line with all other studied languages. We proposed a number of ways to support developers in commenting their Simulink models. This could be done by modifying, or adding Simulink IDE features, greatly extending guidelines on Simulink comments, and comment smell detection – we expect many of our suggestions to also be useful for other visual languages and their tools.

While our work only learns from artifacts, the models and source code, in the future, we want to directly survey developers. Receiving opinions on how developers intend to document, their thought process while doing so, and their struggles, would put our findings into a more complete perspective. Similarly, we could scrape Simulink documentation related discussion from forums or mailing lists, like in [37], to gather insights into Simulink-specific documentation issues.

As the current guidelines on MATLAB and Simulink commentary are leaving many gaps and are not widely followed, we would like to create exhaustive modeling guidelines together with practitioners. This would be particularly useful in partnership with an industrial partner, as our current knowledge only comes from open source projects. After guideline synthesis, we plan to build a comment smell detector, which finds parts that need (more) commentary or even automatically refactors them.

References

[1] Silvia Abrahão, Francis Bourdeleau, Betty Cheng, Sahar Kokaly, Richard Paige, Harald Stöerrle, and Jon Whittle, *User experience for model-driven engineering: Challenges and future directions*, 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2017, pp. 229–236.

[2] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C Shepherd, *Software documentation: the practitioners' perspective*, Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, 2020, pp. 590–601.

[3] Tiago Amorim, Alexander Boll, Ferry Bachmann, Timo Kehrer, Andreas Vogelsang, and Hartmut Pohlheim, *Simulink bus usage in practice: an empirical study*, Journal of Object Technology **22** (2023), no. 2, 2:1–14, The 19th European Conference on Modelling Foundations and Applications (ECMFA 2023).

[4] Paul Barnard, *Software development principles applied to graphical model development*, AIAA Modeling and Simulation Technologies Conference and Exhibit, 2005, p. 5888.

[5] Arianna Blasi, Nataliia Stulova, Alessandra Gorla, and Oscar Nierstrasz, *Replicomment: Identifying clones in code comments*, Journal of Systems and Software **182** (2021), 111069.

[6] Barry W Boehm, John R Brown, and Myron Lipow, *Quantitative evaluation of software quality*, Proceedings of the 2nd international conference on Software engineering, 1976, pp. 592–605.

[7] Alexander Boll, Florian Brokhausen, Tiago Amorim, Timo Kehrer, and Andreas Vogelsang, *Characteristics, potentials, and limitations of open-source Simulink projects for empirical research*, Software and Systems Modeling **20** (2021), no. 6, 2111–2130.

[8] Alexander Boll, Nicole Vieregg, and Timo Kehrer, *Replicability of experimental tool evaluations in model-based software and systems engineering with MATLAB/Simulink*, Innovations in Systems and Software Engineering (2022), 1–16.

[9] Raymond PL Buse and Westley R Weimer, *Learning a metric for code readability*, IEEE Transactions on software engineering **36** (2009), no. 4, 546–558.

[10] Shafiul Azam Chowdhury, Lina Sera Varghese, Soumik Mohian, Taylor T Johnson, and Christoph Csallner, *A curated corpus of Simulink models for model-based empirical studies*, Proceedings of the 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems, 2018, pp. 45–48.

[11] Moacyr AG De Brito, Leonardo P Sampaio, G Luigi, Guilherme A e Melo, and Carlos A Canesin, *Comparative analysis of MPPT techniques for PV applications*, 2011 International Conference on Clean Electrical Power (ICCEP), IEEE, 2011, pp. 99–104.

[12] James L Elshoff and Michael Marcotty, *Improving computer program readability to aid modification*, Communications of the ACM **25** (1982), no. 8, 512–521.

[13] Jay Graylin, Joanne E Hale, Randy K Smith, Hale David, Nicholas A Kraft, WARD Charles, et al., *Cyclomatic complexity and lines of code: empirical evidence of a stable linear relationship*, Journal of Software Engineering and Applications **2** (2009), no. 03, 137.

[14] Alireza Haghighatkhah, Ahmad Banijamali, Olli-Pekka Pakanen, Markku Oivo, and Pasi Kuvaja, *Automotive software engineering: A systematic mapping study*, Journal of Systems and Software **128** (2017), 25–55.

[15] Hao He, *Understanding source code comments at large-scale*, Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019, pp. 1217–1219.

[16] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin, *Deep code comment generation*, Proceedings of the 26th conference on program comprehension, 2018, pp. 200–210.

[17] Yuan Huang, Hanyang Guo, Xi Ding, Junhuai Shu, Xiangping Chen, Xiapu Luo, Zibin Zheng, and Xiaocong Zhou, *A comparative study on method comment and inline comment*, ACM Transactions on Software Engineering and Methodology **32** (2023), no. 32, 1–26.

[18] Yuan Huang, Nan Jia, Junhuai Shu, Xinyu Hu, Xiangping Chen, and Qiang Zhou, *Does your code need comment?*, Software: Practice and Experience **50** (2020), no. 3, 227–245.

[19] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer, *Summarizing source code using a neural attention model*, 54th Annual Meeting of the Association for Computational Linguistics 2016, Association for Computational Linguistics, 2016, pp. 2073–2083.

[20] Monika Jaskolka, Vera Pantelic, Alan Wassyng, Mark Lawford, and Richard Paige, *Repository mining for changes in Simulink models*, 2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2021, pp. 46–57.

[21] Marcel Jerzyk and Lech Madeyski, *Code smells: A comprehensive online catalog and taxonomy*, Developments in Information and Knowledge Management Systems for Business Applications: Volume 7, Springer, 2023, pp. 543–576.

[22] Ninus Khamis, René Witte, and Juergen Rilling, *Automatic quality assessment of source code comments: the JavadocMiner*, Natural Language Processing and Information Systems: 15th International Conference on Applications of Natural Language to Information Systems, NLDB 2010, Cardiff, UK, June 23-25, 2010. Proceedings 15, Springer, 2010, pp. 68–79.

[23] Marija Kostić, Vuk Batanović, and Boško Nikolić, *Monolingual, multilingual and cross-lingual code comment classification*, Engineering Applications of Artificial Intelligence **124** (2023), 106485.

[24] Marija Kostić, Aleksa Srbljanović, Vuk Batanović, and Boško Nikolić, *Code comment classification taxonomies*, Proceedings of the Ninth IcETRAN Conference, 2022.

[25] Fred T Krogh, *Algorithms policy*, ACM Transactions on Mathematical Software (TOMS) **4** (1978), no. 2, 97–99.

[26] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson, *Assessing the state-of-practice of model-based engineering in the embedded systems domain*, Model-Driven Engineering Languages and Systems (Cham) (Juergen Dingel, Wolfram Schulte, Isidro Ramos, Silvia Abrahão, and Emilio Insfran, eds.), Springer International Publishing, 2014, pp. 166–182.

[27] Bo Lin, Shangwen Wang, Zhongxin Liu, Xin Xia, and Xiaoguang Mao, *Predictive comment updating with heuristics and ast-path-based neural learning: A two-phase approach*, IEEE Transactions on Software Engineering **49** (2022), no. 4, 1640–1660.

[28] Shifan Liu, Zhanqi Cui, Xiang Chen, Jun Yang, Li Li, and Liwei Zheng, *Tbcup: A transformer-based code comments updating approach*, 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE, 2023, pp. 892–897.

[29] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke, *On the comprehension of program comprehension*, ACM Transactions on Software Engineering and Methodology (TOSEM) **23** (2014), no. 4, 1–37.

[30] Rajib Mall, *Fundamentals of software engineering*, PHI Learning Pvt. Ltd., 2018.

[31] Vishal Misra, Jakku Sai Krupa Reddy, and Sridhar Chimalakonda, *Is there a correlation between code comments and issues? an exploratory study*, Proceedings of the 35th Annual ACM Symposium on Applied Computing, 2020, pp. 110–117.

[32] Eriko Nurvitadhi, Wing Wah Leung, and Curtis Cook, *Do class comments aid Java program understanding?*, 33rd Annual Frontiers in Education, 2003. FIE 2003., vol. 1, IEEE, 2003, pp. T3C–T3C.

[33] Vera Pantelic, Alexander Schaap, Alan Wassyng, Victor Bandur, and Mark Lawford, *Something is rotten in the state of documenting Simulink models.*, MODELSWARD, 2019, pp. 503–510.

[34] Luca Pascarella and Alberto Bacchelli, *Classifying code comments in Java open-source software systems*, 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), 2017, pp. 227–237.

[35] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo, *Categorizing the content of GitHub readme files*, Empirical Software Engineering **24** (2019), 1296–1327.

[36] Pooja Rani, Suada Abukar, Nataliia Stulova, Alexandre Bergel, and Oscar Nierstrasz, *Do comments follow commenting conventions? a case study in Java and Python*, 2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE, 2021, pp. 165–169.

[37] Pooja Rani, Mathias Birrer, Sebastiano Panichella, Mohammad Ghafari, and Oscar Nierstrasz, *What do developers discuss about code comments?*, 2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE, 2021, pp. 153–164.

[38] Pooja Rani, Arianna Blasi, Nataliia Stulova, Sebastiano Panichella, Alessandra Gorla, and Oscar Nierstrasz, *A decade of code comment quality assessment: A systematic literature review*, Journal of Systems and Software **195** (2023), 111515.

[39] Pooja Rani, Sebastiano Panichella, Manuel Leuenberger, Andrea Di Sorbo, and Oscar Nierstrasz, *How to identify class comment types? a multi-language approach for class comment classification*, Journal of Systems and Software **181** (2021), 111047.

[40] Pooja Rani, Sebastiano Panichella, Manuel Leuenberger, Mohammad Ghafari, and Oscar Nierstrasz, *What do class comments tell us? an investigation of comment evolution and practices in pharo smalltalk*, Empirical Software Engineering **26** (2021), no. 6, 112.

[41] Jef Raskin, *Comments are more important than code: The thorough use of internal documentation is one of the most-overlooked ways of improving software quality and speeding implementation.*, Queue **3** (2005), no. 2, 64–65.

[42] Alexander Schaap, Gordon Marks, Vera Pantelic, Mark Lawford, Gehan Selim, Alan Wassyng, and Lucian Patcas, *Documenting Simulink designs of embedded systems*, Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (New York, NY, USA), MODELS '18, Association for Computing Machinery, 2018, p. 47–51.

[43] Jan Schroeder, Christian Berger, Miroslaw Staron, Thomas Herpel, and Alessia Knauss, *Unveiling anomalies and their impact on software quality in model-based automotive software revisions with software metrics and domain experts*, Proceedings of the 25th International Symposium on Software Testing and Analysis, 2016, pp. 154–164.

[44] Christoph Daniel Schulze, Christina Plöger, and Reinhard von Hanxleden, *On comments in visual languages*, Diagrammatic Representation and Inference: 9th International Conference, Diagrams 2016, Philadelphia, PA, USA, August 7-10, 2016, Proceedings 9, Springer, 2016, pp. 219–225.

[45] Sohil Lal Shrestha, Alexander Boll, Shafiul Azam Chowdhury, Timo Kehrer, and Christoph Csallner, *Evosl: A large open-source corpus of changes in Simulink models & projects*, MODELS '23: ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems, 2023, pp. 273–284.

[46] Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner, *Slnet: A redistributable corpus of 3rd-party Simulink models*, Proceedings of the 19th International Conference on Mining Software Repositories (New York, NY, USA), MSR '22, Association for Computing Machinery, 2022, p. 237–241.

[47] Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner, *Replicability study: Corpora for understanding Simulink models & projects*, 2023.

[48] Murali Sridharan, Mika Mäntylä, Maëlick Claes, and Leevi Rantala, *Soccminer: a source code-comments and comment-context miner*, Proceedings of the 19th International Conference on Mining Software Repositories, 2022, pp. 242–246.

[49] Daniela Steidl, Benjamin Hummel, and Elmar Juergens, *Quality analysis of source code comments*, 2013 21st international conference on program comprehension (icpc), Ieee, 2013, pp. 83–92.

[50] Margaret-Anne Storey, Jody Ryall, Janice Singer, Del Myers, Li-Te Cheng, and Michael Muller, *How software developers use tagging to support reminding and refinding*, IEEE Transactions on software engineering **35** (2009), no. 4, 470–483.

[51] Mario F Triola, William Martin Goodman, Richard Law, and Gerry Labute, *Elementary statistics*, Pearson/Addison-Wesley Reading, MA, 2006.

[52] Chao Wang, Hao He, Uma Pal, Darko Marinov, and Minghui Zhou, *Suboptimal comments in Java projects: From independent comment changes to commenting practices*, ACM Transactions on Software Engineering and Methodology **32** (2023), no. 2, 1–33.

[53] Jens Weiland and Peter Manhart, *A classification of modeling variability in Simulink*, Proceedings of the 8th international workshop on variability modelling of software-intensive systems, 2014, pp. 1–8.

[54] Fengcai Wen, Csaba Nagy, Gabriele Bavota, and Michele Lanza, *A large-scale empirical study on code-comment inconsistencies*, 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), IEEE, 2019, pp. 53–64.

[55] Rebecca Wirfs-Brock and Alan McKean, *Object design: roles, responsibilities, and collaborations*, Addison-Wesley Professional, 2003.

[56] Scott N Woodfield, Hubert E Dunsmore, and Vincent Y Shen, *The effect of modularization and comments on program comprehension*, Proceedings of the 5th international conference on Software engineering, 1981, pp. 215–223.

[57] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li, *Measuring program comprehension: A large-scale field study with professionals*, IEEE Transactions on Software Engineering **44** (2018), no. 10, 951–976.

[58] Jingyi Zhang, Lei Xu, and Yanhui Li, *Classifying Python code comments based on supervised learning*, Web Information Systems and Applications (Cham) (Xiaofeng Meng, Ruixuan Li, Kanliang Wang, Baoning Niu, Xin Wang, and Gansen Zhao, eds.), Springer International Publishing, 2018, pp. 39–47.

[59] Xiaowei Zhang, Weiqin Zou, Lin Chen, Yanhui Li, and Yuming Zhou, *Towards the analysis and completion of syntactic structure ellipsis for inline comments*, IEEE Transactions on Software Engineering **49** (2022), no. 4, 2285–2302.