11-2020

Beyond accuracy: Assessing software documentation quality

Christoph TREUDE
Singapore Management University, ctreude@smu.edu.sg

Justin MIDDLETON

Thushari ATAPATTU

Citation

Beyond Accuracy: Assessing Software Documentation Quality

Christoph Treude
christoph.treude@adelaide.edu.au
University of Adelaide
Adelaide, SA, Australia

Justin Middleton
jamiddl2@ncsu.edu
North Carolina State University
Raleigh, NC, United States

Thushari Atapattu
thushari.atapattu@adelaide.edu.au
University of Adelaide
Adelaide, SA, Australia

ABSTRACT

Good software documentation encourages good software engineering, but the meaning of "good" documentation is vaguely defined in the software engineering literature. To clarify this ambiguity, we draw on work from the data and information quality community to propose a framework that decomposes documentation quality into ten dimensions of structure, content, and style. To demonstrate its application, we recruited technical editors to apply the framework when evaluating examples from several genres of software documentation. We summarise their assessments—for example, reference documentation and README files excel in quality whereas blog articles have more problems—and we describe our vision for reasoning about software documentation quality and for the expansion and potential of a unified quality framework.

CCS CONCEPTS

• **Software and its engineering** → **Documentation**; • **Social and professional topics** → **Quality assurance**.

KEYWORDS

software documentation, quality

1 INTRODUCTION AND MOTIVATION

High-quality software documentation is crucial for software development, comprehension, and maintenance, but the ways that documentation can suffer poor quality are numerous. For example, Aghajani et al. [1] have designed a taxonomy of 162 documentation issue types, covering information content, presentation, process-related matters, and tool-related matters. Additionally, because software documentation is written in informal natural language which is inherently ambiguous, imprecise, unstructured, and complex in syntax and semantics, its quality can often only be evaluated

manually [6]. The assessment depends on context for some quality attributes (e.g., task orientation, usefulness), it mixes content and medium for others (e.g., visual effectiveness, retrievability), and sometimes it requires looking beyond the documentation itself (e.g., accuracy, completeness). While most related work has focused on documentation accuracy and completeness (e.g., for API documentation [3, 19]), the fact that many developers are reluctant to carefully read documentation [20] suggests that documentation suffers from issues beyond accuracy. As such, we lack a comprehensive framework and instruments for assessing software documentation quality beyond these characteristics.

In this work we adapt quality frameworks from the data and information quality community to the software engineering domain. We design a survey instrument to assess software documentation quality from different sources. A pilot study with four technical editors and 41 documents related to the R programming language provides initial evidence for the strengths and weaknesses of different genres of documentation (blog articles, reference documentation, README files, Stack Overflow threads, tutorials) based on the ten dimensions of our software documentation quality framework.

The contributions of this work are:

- A ten-dimensional framework for asking questions about software documentation quality,
- A partially validated survey instrument to evaluate document quality over multiple documentation genres, and
- A vision for the expansion of a unified quality framework through further experimentation.

2 BACKGROUND AND RELATED WORK

The most related piece of work to this paper is the seminal 1995 article "Beyond Accuracy: What Data Quality Means to Data Consumers" by Wang and Strong [17]. We follow the same *beyond-accuracy* approach for the domain of software documentation.

Defective Software Documentation. Defect detection tools have been widely investigated at the code level, but very few studies focus on defects at the document level [22]. The existing approaches in the documentation space investigate inconsistencies between code and documentation. In one of the first such attempts, Tan et al. [16] presented @tcomment for testing Javadoc comments related to null values and exceptions. DocREF by Zhong and Su [19] detects API documentation errors by seeking out mismatches between code names in natural-language documentation and code. AdDoc by Dagenais and Robillard [3] automatically discovers documentation patterns which are defined as coherent sets of code elements that are documented together. Also aimed at inconsistencies between code and documentation, Ratol and Robillard [15] presented Fraco, a tool to detect source code comments that are fragile with respect to identifier renaming.

Wen et al. [18] presented a large-scale empirical study of code-comment inconsistencies, revealing causes such as deprecation and refactoring. Zhou et al. [21, 22] contributed a line of work on detecting defects of API documents with techniques from program comprehension and natural language processing. They presented DRONE to automatically detect directive defects and recommend solutions to fix them. And in recent work, Panthaplackel et al. [8] proposed an approach that learns to correlate changes across two distinct language representations, to generate a sequence of edits that are applied to existing comments to reflect code modifications.

To the best of our knowledge, little of the existing work targets the assessment or improvement of the quality of software documentation beyond accuracy and completeness; one example is Plösch and colleagues' survey on what developers value in documentation [13]. We turn to related work from the data and information quality community to further fill this gap.

Information Quality. In their seminal 1995 work "Beyond Accuracy: What Data Quality Means to Data Consumers", Wang and Strong [17] conducted a survey to generate a list of data quality attributes that capture data consumers' perspectives. These perspectives were grouped into accuracy, relevancy, representation, and accessibility, and they contained a total of 15 items such as believability, reputation, and ease of understanding. A few years later, Eppler [4] published a similar list of dimensions which contained 16 dimensions such as clarity, conciseness, and consistency.

These two sets of dimensions built the starting point into our investigation of dimensions from the data and information quality community which might be applicable to software documentation. Owing to the fact that software documentation tends to be disseminated over the Internet, we further included the work of Knight and Burn [7], who discussed the development of a framework for assessing information quality on the World Wide Web and compiled a list of 20 common dimensions for information and data quality from previous work, including accuracy, consistency, and timeliness.

3 SOFTWARE DOCUMENTATION QUALITY

Inspired by the dimensions of information and data quality identified in related work, we designed the software documentation quality framework summarised in Table 1. The first and last author of this paper collaboratively went through the dimensions from related work cited in Section 2 to select those that (i) could apply to software documentation, (ii) do not depend on the logistics of accessing the documentation (e.g., security), and (iii) can be checked independently of other artefacts (i.e., not accuracy or completeness). As a result, we omitted dimensions such as relevancy and timeliness from Wang and Strong's work [17] since they require context beyond the documentation itself (e.g., relevancy cannot be assessed without a specific task in mind); and we omitted dimensions such as interactivity and speed from Eppler's work [4] which are characteristics of the medium rather than the content. Table 1 indicates the sources from related work for each dimension.

In addition, we formulated questions that an assessor can answer about a piece of software documentation to indicate its quality in each dimension. Our pilot study in Section 4 provides initial evidence that technical editors are able to answer these questions and shows interesting trends across genres of documentation.

The questions were inspired by work on readability by Pitler and Nenkova [12] who collected readability ratings using four questions: How well-written is this article? How well does the text fit together? How easy was it to understand? How interesting is this article? Our framework contains the same questions, and complements them with one question each for the additional six dimensions.

4 PILOT STUDY

To gather preliminary evidence on whether experts would be able to use the proposed framework for assessing software documentation, we conducted a pilot study with technical editors and different genres in software documentation.

Recruitment. To recruit technical editors, we posted an advertisement on Upwork,[1] a website for freelance recruitment and employment. Our posting explained our goal of evaluating software documentation on ten dimensions and identifying weaknesses in their design. Therefore, we required applicants to be qualified as technical editors with programming experience, and accordingly, we would compensate them hourly according to their established rates. From this, we recruited the first four qualified applicants with experience in technical writing and development. We refer to our participants henceforth as E1 through E4.

Data Collection. Because developers write software instructions in a variety of contexts, we use a broad definition of documentation, accepting both official, formal documentation and implicit documentation that emerges from online discussion. We focus on the genres of software documentation that have been the subject of investigation in previous work on software documentation: reference documentation [5], README files [14], tutorials [11], blogs [10], and Stack Overflow threads [2]. To better control for variation between language, we focused exclusively on resources documenting the R programming language or projects built with it. All documentation was then randomly sampled from the following sources:

- **Reference Documentation (RD)**: from the 79 subsections of the R language manual.[2]
- **README files (R)**: from the first 159 items in a list[3] of open-source R projects curated by GitHub users.
- **Tutorials (T)**: from the 46 R tutorials on tutorialspoint.com.[4]
- **Articles (A)**: from 1,208 articles posted on R-bloggers.com,[5] a blog aggregation website, within the last six months.
- **Stack Overflow threads (SO)**: from questions tagged "R".

We asked the editors to read the documents, suggest edits, and answer the ten questions listed in Table 1 using a scale from 1 (low) to 10 (high). From a methodological standpoint, we asked for edits in addition to the ten assessments in order to encourage and evidence reflection on the specific ways the documentation failed, beyond the general impressions of effectiveness. These edits were collected through Microsoft Word's Review interface. We assigned the quantity and selection of documentation according to the amount of time editors had available while trying to balance

---

[1] https://www.upwork.com/
[2] https://cran.r-project.org/doc/manuals/r-release/R-intro.html
[3] https://github.com/qinwf/awesome-R, the remaining items were not software projects
[4] https://www.tutorialspoint.com/r/index.htm
[5] https://www.r-bloggers.com/

Table 1: Dimensions of software documentation quality

Dimension	Question	Source
Quality	How well-written is this document (e.g., spelling, grammar)?	question from [12]
Appeal	How interesting is it?	question from [12]
Readability	How easy was it to read?	'accessibility' [4, 7]
Understandability	How easy was it to understand?	'ease of understanding' [17]; question from [12]
Structure	How well-structured is the document?	'navigation' from [7]
Cohesion	How well does the text fit together?	question from [12]
Conciseness	How succinct is the information provided?	'concise', 'amount of data' [7, 17]; 'conciseness' [4]
Effectiveness	Does the document make effective use of technical vocabulary?	'vocabulary' [12]
Consistency	How consistent is the use of terminology?	'consistency' [4, 7]
Clarity	Does the document contain ambiguity?	'clarity' [4]; 'understandability' [7]

Table 2: Distribution of documentation genres to editors

Editor	RD	R	T	A	SO	Sum
E1	7	7	7	8	7	36
E2	6	6	5	6	6	29
E3	1	1	1	1	1	5
E4	1	1	1	1	1	5
Sum	15	15	14	16	15	75
Union	8	8	8	9	8	41

Table 3: Results of document rating by technical editors

Dimension	RD	R	T	A	SO
Quality	**8.1**	7.6	7.5	*6.1*	6.7
Appeal	5.7	**6.5**	6.4	*5.3*	5.8
Readability	**6.6**	6.4	6.5	*4.6*	6.1
Understandability	6.9	**7.0**	6.9	*5.3*	6.1
Structure	6.3	**7.6**	7.1	*3.8*	5.8
Cohesion	6.7	6.7	**7.0**	*4.9*	6.1
Conciseness	6.7	6.7	**6.8**	*5.3*	6.3
Effectiveness	**8.3**	7.9	8.1	*7.1*	7.5
Consistency	**9.3**	8.8	9.0	*8.5*	8.6
Clarity	8.6	**8.7**	7.6	*6.3*	7.7

distribution across genres. Furthermore, we tried to give each individual document to at least two editors. We show the distribution in Table 2. Overall, the editors worked on 41 documents. Seven documents were assessed by only one editor, but the rest were by two; therefore, we had 75 records of document evaluations.

Results. We display the average dimension rating per genre of documentation in Table 3; for each row, the highest value is in bold, and the lowest is italicised. For example, this table shows that the quality of the reference documentation, or how well-written it is according to the definition in Table 1, is rated 8.1 out of 10, on average higher than all the other genres. Meanwhile, the average blog article was rated 6.1 out of 10, the lowest. Within dimensions, the highest ratings are distributed primarily among reference documentation and README files; tutorials received the highest in cohesion and conciseness, but only by a difference of at most 0.3 of a rating point with the two aforementioned genres. Nevertheless, blog articles received the global lowest rating across every dimension, especially so in readability and structure.

Different dimensions do not seem to have equal distributions. For example, reference documentation has a 9.3 in consistency, but the lowest score is an 8.5 for blog articles. These are nevertheless high scores on average. Appeal also has a small spread, from 6.5 in README files to 5.3 in blog articles, but a lower average overall across dimensions. Structure, on the other hand, varies from 7 (README files) to 3.8 (blog articles), demonstrating a larger interaction with genres.

Reflection. Our findings suggest a trend in rating samples from certain genres highly while disapproving others. The relatively

high rating of reference documentation and README files may be explained by their typical origins within the software product and team, whereas blog articles and Stack Overflow posts can be published more freely [9]. Nevertheless, the result that blog articles received the lowest in every category did surprise us, especially as we designed the framework to reduce the impact of context. There may be many factors involved in these results. For one, the corpora from which we randomly sampled may have different underlying distributions on our dimensions. Our reference documentation was sampled from larger related products, whereas manual inspection of the randomly sampled blog articles did evidence a broad variety of R-related topics, such as discussions of R package updates or write-ups and reflections on personal projects. Furthermore, as with any participant study, the soundness of our results depends on the accuracy with which we communicated our ideas and the participants understood and enacted them. Editors may approach different genres with different preconceptions; an 8 rating for a README file may not be the same as an 8 rating for a Stack Overflow thread.

Technical Challenges. As noted previously, we asked editors to make edits on document copies in Microsoft Word as well, hoping to gain insight into technical editing strategies for software documentation. We obtained over 4,000 edit events across all 75 documents, ranging from reformattings and single character insertions to deep paragraph revisions. However, we encountered several obstacles when attempting to analyse this data. First, when copying online

documents to Microsoft Word, interactions and images in hypertext documents were distractingly reformatted for a linear document. As a result, several of the edits addressed changes to structure that did not exist when viewing the document in the browser instead. Furthermore, many edits were not recorded as intended—for example, replacing words with synonyms that do not make sense in a programming context. Although we can recreate changes to the document text through before-and-after comparison, small edits blend together and large edits confound the differencing algorithm. Because the editor's original intent is lost, any coding scheme applied over it becomes less secure. Due to these circumstances, we decided against drawing conclusions from this data.

5 IMPACT AND FUTURE WORK

Accuracy of software documentation is a necessary but not sufficient condition for its success. To empower software developers to make effective use of software documentation, it must be carefully designed in appearance, content, and more. Our vision, then, is to provide a more precise definition to better explore and transform what it means for software documentation to be of high quality, along with a research agenda enabled by such a quality framework. To that end, we have drawn from seminal work in the information and data quality communities to design a framework consisting of ten dimensions for software documentation quality.

Furthermore, our research agenda proposes future work to evaluate the impact of each dimension with end-users and improve the framework. For one, we can verify the assessments of technical editors by introducing end-users to different versions of the documents (e.g., edited by technical editors based on quality criteria and original version) and observing their use. Another direction is to explore trade-offs between quality attributes, such as whether readability outweighs structure in terms of document usability, and to further disambiguate similar dimensions (e.g., structure and cohesion). We will further revisit the edits from the technical editors to extract surface-level (e.g., word count) and deep-level (e.g., cohesion of adjacent sentences) lexical, syntactic, and discourse features to build classification models for predicting documentation quality. Such classification models can be used to assess and rank the quality of software documentation.

We believe these efforts are important because of the volume of software documentation on the web. A simple Google search related to software development will return documentation that matches the query without explicitly considering the quality of the material. For example, a query on 'reading CSV file in r' returns blog articles and tutorials as the top results, yet our preliminary results demonstrate that on average, blog articles are ranked worst across all ten quality dimensions. This is not to say that blog articles are essentially faulty and should be abandoned moving forward; rather, we hope to spur reflection on how end-users interact with blog articles and how each of our dimensions manifest uniquely under the genre's constraints, while nevertheless using what we currently know to emphasise more useful results. Therefore, applying the framework can influence guidelines and recommendation systems for documentation improvement as well as automatic assessing and ranking systems for navigating the large volumes of documentation and emphasising high-quality documentation.

REFERENCES

[1] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software documentation issues unveiled. In *Int'l. Conf. on Software Engineering*. 1199–1210.

[2] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. 2014. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering* 19, 3 (2014), 619–654.

[3] Barthélémy Dagenais and Martin P Robillard. 2014. Using traceability links to recommend adaptive changes for documentation evolution. *IEEE Trans. on Software Engineering* 40, 11 (2014), 1126–1146.

[4] Martin J Eppler. 2001. A generic framework for information quality in knowledge-intensive processes. In *Int'l. Conf. on Information Quality*.

[5] Davide Fucci, Alireza Mollaalizadehbahnemiri, and Walid Maalej. 2019. On using machine learning to identify knowledge in API reference documentation. In *Joint Meeting on European Software Engineering Conference and Symp. on the Foundations of Software Engineering*. 109–119.

[6] Ninus Khamis. 2013. *Applying Computational Linguistics on API Documentation: The JavadocMiner*. LAP LAMBERT Academic Publishing.

[7] Shirlee-ann Knight and Janice Burn. 2005. Developing a framework for assessing information quality on the World Wide Web. *Informing Science* 8 (2005).

[8] Sheena Panthaplackel, Pengyu Nie, Milos Gligoric, Junyi Jessy Li, and Raymond J Mooney. 2020. Learning to Update Natural Language Comments Based on Code Changes. *arXiv preprint arXiv:2004.12169* (2020).

[9] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. 2012. *Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow*. Technical Report. Georgia Institute of Technology.

[10] Chris Parnin, Christoph Treude, and Margaret-Anne Storey. 2013. Blogging developer knowledge: Motivations, challenges, and future directions. In *Int'l. Conf. on Program Comprehension*. 211–214.

[11] Gayane Petrosyan, Martin P Robillard, and Renato De Mori. 2015. Discovering information explaining API types using text classification. In *Int'l. Conf. on Software Engineering*. 869–879.

[12] Emily Pitler and Ani Nenkova. 2008. Revisiting readability: A unified framework for predicting text quality. In *Conference on Empirical Methods in Natural Language Processing*. 186–195.

[13] Reinhold Plösch, Andreas Dautovic, and Matthias Saft. 2014. The value of software documentation quality. In *Int'l. Conf. on Quality Software*. 333–342.

[14] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the content of GitHub README files. *Empirical Software Engineering* 24, 3 (2019), 1296–1327.

[15] Inderjot Kaur Ratol and Martin P Robillard. 2017. Detecting fragile comments. In *Int'l. Conf. on Automated Software Engineering*. 112–122.

[16] Shin Hwei Tan, Darko Marinov, Lin Tan, and Gary T Leavens. 2012. @tcomment: Testing javadoc comments to detect comment-code inconsistencies. In *Int'l. Conf. on Software Testing, Verification and Validation*. 260–269.

[17] Richard Y Wang and Diane M Strong. 1996. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems* 12, 4 (1996), 5–33.

[18] Fengcai Wen, Csaba Nagy, Gabriele Bavota, and Michele Lanza. 2019. A large-scale empirical study on code-comment inconsistencies. In *Int'l. Conf. on Program Comprehension*. 53–64.

[19] Hao Zhong and Zhendong Su. 2013. Detecting API documentation errors. In *Int'l. Conf. on Object-oriented Programming Systems Languages, and Applications*. 803–816.

[20] Hao Zhong, Lu Zhang, Tao Xie, and Hong Mei. 2009. Inferring resource specifications from natural language API documentation. In *Int'l. Conf. on Automated Software Engineering*. 307–318.

[21] Yu Zhou, Changzhi Wang, Xin Yan, Taolue Chen, Sebastiano Panichella, and Harald C Gall. 2018. Automatic detection and repair recommendation of directive defects in Java API documentation. *IEEE Trans. on Software Engineering* (2018).

[22] Yu Zhou, Xin Yan, Taolue Chen, Sebastiano Panichella, and Harald Gall. 2019. DRONE: a tool to detect and repair directive defects in Java APIs documentation. In *Int'l. Conf. on Software Engineering: Companion Proceedings*. 115–118.