Evaluating the Quality of the Documentation of Open Source Software

Lerina Aversano, Daniela Guardabascio and Maria Tortorella
Department of Engineering, University of Sannio, p.za Roma 21, Benevento, Italy

Keywords: Open Source Software, Software Maintenance, Software Reusability, Software Measurement, Software Metrics, Software Documentation.

Abstract: Software documentation is a basic component of the software development process: from the definition of the functional requirements to the maintenance phase. Software documentation refers to different types of documents that facilitates the software developer's tasks. Then, it includes the textual documentation required by the Software engineering standards, API documentation, Wiki pages and source code comments. Surveys and studies indicate that the documentation is not always available and, if available, only partially addresses the developers' needs, as it is often wrong, incomplete, out-of-date and ambiguous. This paper focuses on the quality assessment of the documentation of open source systems with the aim of understanding the support it can offer for adopting them and executing maintenance activities. Specifically, a quality model is defined and a preliminary investigation of its applicability is performed.

1 INTRODUCTION

Software documentation has a significant relevance in the software development process from the definition of the functional requirements to the maintenance phase (Chomas and Saini, 2015), (Garousi et al., 2013), (Sommerville, 2005).

Actually, the software is not only documented by the textual documentation required by the software engineering standards (IEEE Std 830-1998), (IEEE Std 1028-2008), (IEEE Std 1063-2001), (ISO/IEC 9126:2001), (ISO/IEC 25010:2005), (ISO/IEC 26514:2008), even if in different formats (e.g., Word, pdf, ppt), but it can include additional documents describing the software artefacts, such as all the documents produced during the software development process, and formalized as API documentation, Wiki pages, and source code comments.

In the proposed study, the term "software documentation" is used to refer the various types of documents indicated above.

The documentation required by the standards (IEEE Std 830-1998), (IEEE Std 1028-2008), (IEEE Std 1063-2001), (ISO/IEC 9126:2001), (ISO/IEC 25010:2005), (ISO/IEC 26514:2008) generally consists of documents aiming to explain the functionalities the software performs, its architecture, how it is structured and implemented, and how it can be used. It includes the following documents: software requirements specifications, software design documents, code, quality and testing documents. Each document is relevant for understanding the software product. Differently, the API documentation specifies how software components can be used and interacts with each other. Wiki pages allow for web-based visualization and knowledge management. It offers semantic-enhanced search facilities such as filtering, faceting, and graph-like exploration of knowledge. Finally, software documentation also includes the inline comments of the source code. Actually, according to de Souza at al. (Cozzetti de Souza et al., 2005), the comments help developers to fully understand the software product.

Several interviews with software engineers and developers working in organizations have been performed. The results are enclosed in surveys and papers, and indicate that the documentation is not always available and it only partially addresses the developers' needs, as it is often wrong, incomplete, out-of-date and ambiguous.

In the case of closed source software, the requirements are generally clear and well-documented, as well as the design and testing

Aversano, L., Guardabascio, D. and Tortorella, M.
Evaluating the Quality of the Documentation of Open Source Software.
DOI: 10.5220/0006369403080313
In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2017)*, pages 308-313
ISBN: 978-989-758-250-9

documents (Satzinger et al., 2000). These does not happen for open source systems, where these documents may not exist, or not represent any official documentation.

This paper focuses on the assessment of the quality of the various types of documents that may be useful for understanding a software artefact. In particular, it is focused on the documentation of the Open Source Software systems, with the aim of understanding if it can be a valuable support to anyone who wants to adopt such a kind of systems and/or execute maintenance activities.

With this in mind, the paper presents a quality model and the results of a preliminary investigation of its applicability.

This paper is organized as follows. Section 2 discusses the principal research work related to the quality of software documentation. Section 3 describes the proposed model for evaluating the quality of the documentation of Open Source Software system. Section 4 presents a case study, while conclusions are given in Section 5.

2 RELATED WORK

The literature reports several studies focusing on the evaluation of the usefulness of the documentation of a software product. Specifically, these studies discuss the use that software practitioners make of different kinds of documentations.

Forward and Lethbridge (Forward and Lethbridge, 2002) made a survey, involving different developers. They presented several documentation attributes, such as document writing style, grammar, level of upgrade, type, format, visibility, etc. They observed that the documentation content is an important support for communication and should always be useful and serve a purpose. It can be relevant even if it is not updated or inconsistent. The same authors (Lethbridge et al., 2003) highlighted the general attitudes regarding the software engineering documentation. For example, some results indicated that various types of abstract documentation are a valid guidance for maintenance work and that inline comments of the source code are often a good support to assist detailed maintenance work. The study also discussed negative results, such as the fact that multiple types of documents are often out of date or that the documentation is poorly written.

de Souza et al. (Cozzetti de Souza et al., 2005) established in their surveys the importance of each documentation artefacts, with reference to the full understanding of the software product and execution of maintenance activities. The results of this study have also shown that the documentation is often incomplete or out-of-date, and the developers have to use the source code and related comments for fully understanding the software product.

Another aspect coming from the contribution of Kipyegen and Korir (Kipyegen and Korir, 2013) regards the little usage of the documentation, and, consequently, the decrease of its efficiency during the software development task.

Several other research works focused on the documentation quality.

Arthur and Stevens (Arthur and Stevens, 1989) identified four Document Quality Indicators (DQI) that are attributed to an appropriate documentation: Accuracy, Completeness, Usability and Expandability. As quality is an intangible characteristic, without direct measure, the authors of the cited paper associated each quality characteristic to some factors, and each factor to some quantifiers, whose combination gave rise to quality indicators (DQI) that are quantifiable.

In (Plösch et al., 2014), (Wingkvist et al., 2010), the following additional documentation quality attributes are proposed: Accuracy, Clarity, Consistency, Readability, Structuring, and Understandability. Indeed, from a conducted survey (Wingkvist et al., 2010), it emerges that the typical problems related to the documentation quality deals with unreliable, incomplete or non-existent documentation, not documented changes in the software system and lack of integrity and coherence. The authors developed a tool (Lethbridge et al., 2003) for analysing the quality of software systems, documentation included.

Problems related to the documentation quality have also been encountered by Uddin and Robillard (Uddin and Robillard, 2015) with reference to the API documentation quality, and by Diaz-Pace et al. (Diaz-Pace et al., 2014) with reference to the Wiki pages.

This paper considers all the aspects arising from the previous studies and proposes a quality model for evaluating various types of documentation associated with an Open Source Software system.

Then, unlike other proposed approaches, focusing on just one kind of documentation, the proposed quality model considers multiple types of document supporting an Open Source Software system, such as textual standard documentation, API documentation, Wiki support and in line comments.

3 QUALITY OF SOFTWARE DOCUMENTATION

This section proposes an approach to assess and evaluate the quality of software documentation. The approach is metric-based and considers different indicators to be evaluated on the various types of documentation considered: technical documentation, user documentation, API, Wiki, and source code comments. The quality indicators have been formalized by using a set of Questions, which entail the evaluation of a set of Metrics. Overall, the identified quality indicators aim to evaluate:

- to what extent the documentation reflects the current state of the software system;
- to what extent the documentation is understandable and well structured.

Tables 1 contains the list of the questions and related metrics that have been considered. For each metric the table reports the formula used for its assessment, the range of values it can assume and its acceptance threshold, if definable. In particular, the acceptance thresholds are useful to identify where the software system documentation needs to be improved, on the basis of the values of the measured metrics. The quality indicators considered for the definition of the metric-based quality model and reported in Table 1 are the following:

- *Completeness*: to evaluate the completeness level of the documentation with reference to the source code; in particular, these quality indicators check whether the documentation describes all of the items (packages, classes, methods) of the source code.
- *Alignment*: to verify whether the documentation is updated with reference to the project release it refers to.
- *Readability:* to examine if the sentences express clear and understandable concepts. The Flesch readability test (Flesch Reading Ease) was used, designed to indicate how difficult is to be understood an English reading. The index score ranges from 0 to 100 and a value equal to 50 is considered adequate as it is associated with the 10th to 12th grade school level; while higher scores indicate material that is easier to read and lower marks indicate passages that are more difficult to read (Flesch and Rudolf, 2016).
- *Dimension:* to analyse if the document sentences are too long or too short, with the aim of verifying if text is too difficult to be understood, or too short for exhaustively expressing a concept, respectively. Cutts asserted that over the whole document, average sentence length should be 15-20 word (Cutts, 2013).
- *Graphical Support:* to verify the availability of visual aids (figures and tables) facilitating the understandability of the text. The citations of the visual aids within the text and existence of clear captions are verified. It may occur that figures or tables not included in the document are indexed, in this case the score is greater than 1.
- *Consistency to Standard*: to verify if the available documentation is compliant to the Software Engineering Standards: Software Requirement specification, Software Design documents, Code system documentation, Test plans, and so on.
- *Structure*: regarding the textual documentation and aiming at evaluating its actual structure in terms of number of chapters, sections, sub-sections nesting, document length, and density of tables and figures. A good structure and organization of the document helps to consult it.
- *Easy to use*: concerning API and Wiki and assessing the organization from a usability point of view. Aspects that will be investigated are the fragmentation of the concepts within web pages, and the size of the Java Doc and Wiki.
- *Appropriateness comments:* estimating the density of comments in the code. A high density of comments helps the developer in the comprehension of source code.

Most of the metrics included in the model can be evaluated with reference to the API, Wiki, code comments and documentation. While Graphical Support and Consistency to Standard are evaluated just with reference to Documentation. Easy to use is evaluated with reference to Wiki and JavaDoc; finally, the Appropriateness of the comments is evaluated with reference to the code comments.

The evaluation of these metrics requires the application of NLP, Information Extraction and Information Retrieval techniques in order to make an objective analysis and not a subjective one. Table 1 shows the metrics considered for the different types of documentation. The index *I* indicates in which set of documents refers analysis, while index *E* refers to what visual aid it is evaluated (figure or table).

4 CASE STUDY

This section described the study conducted to verify the applicability of the proposed metrics-based approach. In particular, the approach has been used

Table 1: Metrics adopted for evaluating the quality of documentation.

QUESTION		METRIC			
Id	Name	Id	Formulas	Values metrics	Acceptance threshold
Q1	What is the completeness level of documentation with reference to the source code?	M1.I	$Completeness_I = \frac{Totl\ classes\ and\ Packages\ described\ in\ I}{Total\ Classes\ and\ Packages\ in\ source\ Code}$ With I ∈ {API, Wiki, Comments, Documentation}	[0..1]	0.6
Q2	Is the available documentation updated with the considered release?	M2.I	$Updating_I$ With I ∈ {API, Wiki, Comments, Documentation}	{yes, not}	Yes
Q3	What is the level of readability of the documentation?	M3.I	$FleschReadabilityIndex_I$ With I ∈ {API, Wiki, Comments, Documentation}	[0..100]	50.0
Q4	What is the medium sentences dimension?	M4.I	$MediumSentenceLenght_I$ With I ∈ {API, Wiki, Comments, Documentation}	Natural number	[15..20]
Q5	If present, the Figures and the Tables are appropriately indexed? Have they a Legend?	M5.1.E	$Number_{D\_E}$ With E ∈ {Figure, Table}	Natural number	
		M5.2.E	$Indexing_{D\_E} = \frac{\#indexed\ figures}{total\ figures}$ With E ∈ {Figure, Table}	[0..1]	0.6
		M5.3.E	$Legend_{D\_E} = \frac{NumberCaptions_{I\_E}}{Total_{I\_E}}$ With E ∈ {Figure, Table}	[0..1]	0.6
Q6	Is the documentation organized in accordance to the standards?	M6.I	$ConsistencySt_D = \frac{\#documents}{\#\ documents\ of\ standards}$	[0..1]	0.6
Q7	Is the documentation well structured?	M7.1	$DocumentSize_D = Number\ pages\ of\ Doc_I$	Natural number	
		M7.2	$AverageSizeCP_D = \frac{\#chapters}{\#pages}$	[0..1]	0.6
		M7.3	$ChapterTreeDept_D$	Natural number	3
		M7.4.E	$Densityy_{D\_E} = \frac{Number_{D\_E}}{TotalPages_{D\_E}}$ With E ∈ {Figure, Table}	[0..1]	0.6
Q8	Is the documentation easy to use?	M8.1	$InfoFragmentation_W = \frac{NumberInfo_W}{NumberPages_W}$	[0..1]	0.6
		M8.2	$JavaDocDensity = \frac{\#classes}{JavaDoc\ MB}$	{0,1,..}	0.8
		M8.3	$WiziSize = \#Wiki\ pages$	Natural number	
		M8.4	$WikiPagesTreeDept$	Natural number	
Q9	Is the code appropriately commented?	M9.1	$CommentDensity = \frac{\#CommenLines}{Lines\ of\ Code}$	[0..1]	0.15

to measure the quality of the documentation of an open source software system, namely, OpenNMS.
OpenNMS (www.opennms.org) is an enterprise network management application platform developed under the Open Source model. This Project started in July of 1999 and was registered on SourceForge in March of 2000. It evolved over seventeen different releases. Currently it includes more than 700 packages.

Table 2 reports the results of the evaluation of the documentation of OpenNMS. A subset of these metrics have been manually computed, such as ChapterTreeDept or DocumentationSize, other with the support of Python scripts, such as Consistency of API and Code Comments.

Table 2: Metrics evaluation of OpenNMS.

Id	NAME	Formulas
M1.D	$Completeness_D$	0
M1.A	$Completeness_A$	0.62
M1.W	$Completeness_W$	0
M1.C	$Completeness_C$	0.29
M2.D	$Updating_D$	Yes
M2.A	$Updating_A$	Yes
M2.W	$Updating_{Wiki}$	Yes
M2.C	$Updating_C$	Yes
M3.D	$FleschReadabilityIndex_D$	43.55
M3.A	$FleschReadabilityIndex_A$	49.48
M3.W	$FleschReadabilityIndex_W$	58.47
M3.C	$FleschReadabilityIndex_C$	53.85
M4.D	$MediumSentenceLenght_D$	12.72 words
M4.A	$MediumSentenceLenght_A$	7.7 words
M4.W	$MediumSentenceLenght_W$	16.73 words
M4.C	$MediumSentenceLenght_C$	8.03 words
M5.1.F	$Number_{D-F}$	56 figures
M5.1.T	$Number_{D-T}$	286 Tables
M5.2.F	$Indexing_{D-F}$	0.0
M5.2.T	$Indexing_{D-F}$	0.0
M5.3.F	$Legend_{D-F}$	0.71
M5.3.T	$Legend_{D-T}$	0.33
M6.I	$ConsistencySt_D$	0.20
M7.1	$DocumentSize_D$	405 pages
M7.2	$AverageSizeCP_D$	11.57 pp/cap
M7.3	$ChapterTreeDept_D$	3 sections
M7.4.F	$Density_{D-F}$	0.14
M7.4.T	$Density_{D-T}$	0.71
M8.1	$InfoFragmention_W$	0.20
M8.2	*JavaDocDensity*	0.19
M8.3	*WikiSize*	2825
M8.4	$WikiPagesTreeDept$	5 subsections
M9.1	*CommentDensity*	0.14

The first indication emerging from the assessment of Question Q1 is that not all the packages and classes in the source code are referred in the API (M1.A), and not all the classes and methods in Source Code are commented (M1.C), indeed both these values are less than one. While Wiki and Documentation have not references to the Source Code, then their Completeness (M1.W, M1.D) is equal to 0.

From the assessment of Question Q2, it emerges that the documentation is updated with reference to the source code. This means that the analysed documentation, API, Wiki and inline comments was aligned with the source code.

The evaluation of Question Q3 indicates that the Documentation (M3.D) and API (M3.A) are difficult to read, and that even the medium sentences length is excessively long (M4.A and M 4.D). Readability of Comments inline of source Code (M3.C) and Readability of Wiki (M3.W) are fairly difficult to be read. Indeed, these metrics assume a value very closed to the thresholds.

Instead, the evaluation of Question Q5 indicates that the documentation includes some figures with clear captions (M5.1.F= 56 figures, M5.3.F = 0.71), while the documentation includes a lot of Tables, but many of them do not have an adequate caption (M5.1.T = 286 tables, M5.3.F = 0.33), or they are not adequately indexed (M5.2.F = 0, M5.2.T = 0).

With reference to the assessment of Question Q6, it is possible to observe that the available documentation is not consistent with the standards. In fact, the documentation includes just one kind of document foreseen in the standards (M6 = 0.20).

With reference to the assessment of Question Q7, it is possible to observe that the number of pages of the documentation is equal to 405 (M7.1), but the documentation analysed includes four documents: administrators guide (264 pages), developers guide (79 pages), installation guide (30 pages), users guide (32 pages). The average length of a chapter is 11.57 pages (M7.2), and the tree depth of each chapter is 3 (M7.3).

In addition, Figure density does not achieve a high value as just 56 figures are included in 405 pages (M7.4.F = 0.13); the reverse can be observed for the Table density as 286 tables are present in 405 pages (M7.4.T = 0.70).

With reference to the assessment of Question Q8, it is possible to observe that the Info Fragmentation does not achieve a high value, then each information is distributed among few pages (M8.1 = 0.20). Regarding JavaDocDensity, about 18 classes are described in one JavaDoc MB indicating that even the Java doc density (M8.2 = 0.18) assumes low values. The size of the Wiki is fairly high (M8.4 = 2825), and has a navigation tree equal to 5 (M8.3 = 5).

Finally, density of comments in the code is fairly low (M9.1 = 0.14), indicating that the code is few commented.

From the obtained results, it can be deduced that the documentation of the reporting software system must be improved. In particular, it can be observed that; the readability needs to be improved; more documents requested by Software Engineering Standards must be included; the comments should be increased within the source code; the API should be aligned to the source code; and Figures and Tables should be indexed.

5 CONCLUSION

This paper proposed a metric-based quality model for the assessment of the quality of the documentation of Open Source Software systems. In particular, the proposed model includes the specification of a set of metrics to be measured. It refers to a wide concept of documentation that considers the various type of documentation that facilitates the software developer's tasks, such as textual documentation required by the Software engineering standards, API documentation, Wiki pages and source code comments.

A preliminary investigation of its applicability of the proposed quality model has been performed by considering an Open Source Software systems, OpenNMS.

Future work will consider a refinement of the quality model with the introduction of additional needed metrics considering grammatical correctness of documentation, ambiguity of the text, duplication of arguments and the analysis of more case studies. Indeed, a larger base of software systems should be measured to increase the practical relevance of the achieved results.

REFERENCES

Arthur, J.D., Stevens, K. T., 1989, Assessing the adequacy of documentation through document quality indicators. *Conference on Software Maintenance, IEEE comp. soc. press*, pp. 40-49.

Briand, L. C., 2003. Software documentation: how much is enough? Software Maintenance and Reengineering, *Seventh European Conference. IEEE comp. soc. press*., pp. 13-15.

Chomas, V. S., Saini, J. R., 2015, Software Template for Evaluating and Scoring Software Project Documentations. *Int. Journal of Computer Applications,* 116(1).

Cozzetti de Souza, S., Anquetil, N., de Oliveira, K.M., 2005. A Study of the Documentation Essential to Software Maintenance. *23rd Annual Int. Conference on Design of communication: documenting & designing for pervasive (SIGDOC '05)*, ACM press, pp.68-75.

Cutts, M., 2013. Oxford guide to plain English. OUP Oxford.

Dautovic A., Plösch, R., Saft, M., 2011. Automated quality defect detection in software development documents. *First Int. Workshop on Model-Driven Software Migration (MDSM 2011).*

Diaz-Pace, J.A., Nicoletti, M., Schettino, S., Grisando, R., 2014. A recommender system for technical software documentation in Wikis. *Biennial Congress of Argentina (ARGENCON),* IEEE comp. soc. press, pp. 393-398.

Garousi, G., Garousi, V., Moussavi, M., Ruhe G., Smith, B., 2013. Evaluating usage and quality of technical software documentation: an empirical study. *17th Int. Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM, pp. 24-35.

Flesch, R., 2016. How to Write Plain English. University of Canterbury.

Forward, A., Lethbridge, T. C., 2002. The relevance of software documentation, tools and technologies: a survey. *Symposium on Document engineering,* ACM press, pp. 26-33.

Kipyegen, N. J., Korir, W. P. K., 2013. Importance of Software Documentation. *IJCSI Int. Journal of Computer Science,* 10(5), pp.1694-0784

IEEE Std 830-1998. 1998. *IEEE Recommended Practice for Software Requirements Specifications*.

IEEE Std 1028-2008. 2008. *IEEE Standard for Software Reviews and Audits*.

IEEE Std 1063-2001. 2001. *IEEE Standard for Software User Documentation.*

ISO/IEC 9126:2001. 2001. *Software engineering – Product quality.* □

ISO/IEC 25010:2005. 2005. *Software engineering – Software product Quality Requirements and Evaluation (SquaRE)*.

ISO/IEC 26514:2008. 2008. *Systems and software engineering - Requirements for designers and developers of user documentation.*

Lethbridge, T. C., Singer, J., Forward, A., 2003. How software engineers use documentation: The state of the practice. *IEEE Software,* 20(6), pp. 35-39.

Plösch, R., Dautovic, A., Saft, M., 2014, The Value of Software Documentation Quality. *14th Int. Conference on Quality Software. IEEE* comp. soc. press, pp. 333-342.

Satzinger, J. W., Jackson, R. B., Burd, S. D., 2000. *System Analysis and Design in a Changing World* , Thomson Learning.

Sommerville I, 2005. Software Documentation. *Software Engineering, Volume 2: The Supporting Process,* pp. 143-154.

Uddin, G., Robillard, M. P., 2015. How API documentation fails. *IEEE Software*, 32(4), pp. 68-75.

Wingkvist, A., Ericsson, M., Lucke, R., Lowe, W., 2010. A metrics-based approach to technical documentation quality. *7th Int. Conference on the Quality of Information and Communications Technology (QUATIC). IEEE comp. soc. press*, pp. 476-481.