Duplicate management in software documentation maintenance

Koznov D.V., Luciv D.V., Chernishev G.A.

Saint Petersburg State University
{d.koznov, d.lutsiv, g.chernyshev}@spbu.ru

Abstract. Software today is becoming increasingly more complex and extensive, and its documentation is becoming more and more complicated. During the software life cycle documentation tends to accumulate a lot of duplicates due to copy-pasting: first, some text fragment is copied, possibly multiple times, then each copy is modified, possibly in its own way; thus, different copies of the initial fragment become *near duplicates*. Depending on the document type, duplicates can be either desired or not. In either case, they need to be managed during documentation lifecycle. This paper introduces a duplicate management process and shows how it could be applied in documentation maintenance. An example of duplicate management for industrial documentation is presented.

1 Introduction

Software documentation has remained an open problem in the software industry for the last 50 years. Brooks reported about it in the early 1970s [1], Somerville wrote the same in the 1990s-2000s [2], Parnas discussed the same problem in 2010s [3]. Scopus shows 4800 papers published between 1969 and 2017, which include "software documentation" as a key word. Research into software documentation is still of high interest today.

One of the important tasks of software documentation development and maintenance is the text fragment duplicate problem [4]. During the software life cycle, the documentation tends to accumulate numerous duplicates due to copy-pasting. First, some text fragment is copied (possibly, more than once), then the copies are modified (possibly, in a number of ways). Thus, various copies of the same text fragment are produced, which become *near duplicates*. There is no single opinion on the duplicates' role in software documentation. For example, Wingkvist et al. [5] consider duplicates as an undesired redundancy of documentation. Oumaziz et al. [6], Koznov et al. [7] and others believe that developers should reuse documentation as much as possible to simplify its maintenance.

Following Parnas [3], we consider a particular type of software documentation, namely *reference documents*: API documents, reference manuals, user guides, etc. These documents aim to help the reader to retrieve specific facts and are not meant to be read from the beginning to the end. Reference documents describe sets of typed

objects such as functions, classes, GUI elements, etc. Consequently, these documents need to be as uniform as possible. This is the reason why they should inevitably include many duplicates. In practice, we deal with real-life reference documents, and to support their unification we need tools for the detection and analysis of existing duplicates. After the duplicates are detected and analyzed, the documentation requires changes towards textual uniformity. During this process, the technical writer corrects the errors and works to increase the clearness, integrity and unambiguity of the documentation.

In our previous studies [8], [9], [10] we suggested an algorithm and a toolkit for near duplicate detection and analysis. However, no conceptual framework for duplicate operation was developed. To close this gap, in the current paper we suggest a duplicate management process and show how it could be applied to reference documentation maintenance. Also, an example of duplicate management for industrial documentation is presented.

2 Related work

Let us consider the approaches which focus on the problem of duplicates in software documentation.

Horie et al. [11] view the problem of document duplicates in Java API documentation. They present a CommentWeaver toolset to support a mechanism for the modularization of the API documentation. The CommentWeaver is implemented as an extension of the Javadoc tool, providing new tags for controlling duplicates. However, near duplicates are not considered, and facilities for duplicate detection are not provided.

Oumaziz et al. [6] perform an empirical study of duplicates in the JavaDoc documentation of seven open source Java-projects (the so called methods documentation). A considerable number of duplicates were detected and there was suggested a classification of duplicates based the on relations of code methods to which documentation duplicates belong. A proposal for an automatic reuse mechanism extending JavaDoc was made. The authors consider partial duplicates in the sense that documentation of the methods may be duplicated partially. However, actually they only consider exact duplicates.

Nosal and Poruban [12] extend the approach from [11] by introducing near duplicates. In this study, the notion of documentation phrase is used to denote the near duplicate.

In [13] Nosal and Poruban present the results of a case study in which they searched for exact duplicates in internal documentation (source code comments) of an open source project set. They used a modified copy-paste detection tool originally developed for code analysis and found a considerable number of text duplicates. However, near duplicates were not considered in this paper.

Wingkvist et al. adapted a clone detection tool to measure document uniqueness in a collection [5]. The authors used the found duplicates for documentation quality estimation. However, they did not address near duplicate detection.

Juergens et al. [4] analyze 28 industrial documents. First, they used a clone detection tool to find duplicates, then filtered the found duplicates by manually removing false positives, and performed a classification of the results. Next, the authors discuss how to use the discovered duplicates and how to detect related duplicates in the source code. The impact of duplicates on the document reading process is also studied. The authors note the presence of near duplicates, but do not take them into account while working with the documentation.

Finally, it should be noted that the problem of duplicates is widely recognized in the software engineering community. Most studies focus on source code documentation and requirements. However, other kinds of documentation are not considered. There is a lack of tool support of reuse techniques, which hinders the practical application of reuse and duplicate management. The real-life practices of documentation maintenance which using duplicate analysis are very poor.

3 Background

In our previous work, we used a token-based code clone detector, Clone Miner toolset [14], for exact duplicate detection. A token in the context of text documents is one single word separated from other words by some separator: '.', '(', ')', etc. Having exact duplicates detected by Clone Miner, we extract sets of duplicate groups where clones are located close to each other. We combine these duplicate groups into a group of near duplicates: every member of this group has one variation to capture different port numbers [10]. Then we apply the adaptive reuse technique [7], [15] to the resulting near duplicates and provide the automatic refactoring of the documentation. Our toolkit automatically creates a parameterized reusable text fragment definition for every near duplicate group. After that, it extracts all occurrences of the group's members from the document, substituting the invocations of the reusable element with the actual parameters.

In [16] we presented a pilot version of a duplicate detection algorithm based on the n-gramm model (a processing natural language processing technique). The algorithm showed some interesting results and appeared quite simple. By continuing this work we hope to overcome the problems of the clone detection approach (low quality of duplicate detected).

4 Duplicate management process

When dealing with the maintenance of reference documents we consider the processes of documentation modification and documentation improvement. When software is modified (new features are added, user interface is changed, etc.), its documentation needs to be modified correspondingly. However, the documentation also requires improvements regardless of software modification demands: it is necessary to correct errors, to restructure and uniform the documentation, to change tools and technologies, etc. For example, Linux Kernal Documentation is actively being improved at the

moment [17]. Below we will give a definition of the duplication management process and show how it can be used for documentation improvement.

Duplicate management is the process of duplicate detection and analysis with the corresponding documentation change to correct errors and provide uniformity, which may or may not include the application of documentation reuse techniques – see fig. 1. Let us briefly consider every phase of the duplicate management process.

The first phase is the automatic duplicate detection of both exact and near duplicates. For that purpose, the following techniques are used [18]: software clone detection and natural language processing (in particular, the n-gramm model).

The second phase is the manual duplicate analysis and documentation changes. On the one hand, automatic duplicate analysis produces significant false positives [4]. Also, the detected duplicates are very often partially correct: they violate text structure and are semantically disclosed [10]. To resolve these problems a human analysis is required.

On the other hand, automatically detected duplicates should be analyzed manually to rewrite the documentation for increasing the level of uniformity and correcting the errors. Automatically detected duplicates provide a lot of significant information for these tasks. After the documentation undergoes changes, more duplicates will be detected.

The third phase is documentation reuse based on the detected duplicates and the performed by documentation changes. In other words, duplicates could be formally reused, as described in [11], [15]. This phase is marked in fig. 1 by a dotted line because, as was mentioned by Oumaziz et al. [6], documentation development frameworks provide insufficient reuse support at the moment. Hence, this function is unavailable for most software development projects and technical writers.

Let us consider in detail how the proposed process can improve documentation quality. For that purpose, we use documentation quality attributes collected in [19], of which we chose *information organization* (document structure), *format* (writing style), and *spelling and grammar*. These attributes are most notably improved by duplicate management.

The first attribute (information organization) is improved as a result of upgrading document structure in the process of duplicate management. Reference documents describe typed objects; consequently, they have uniform sections and subsections. When this is not so, the document structure needs to be redeveloped. Moreover, the structure of the selected sections should be also uniform even if this structure is not supported by headings. For example, when describing any API function we need firstly to give a brief outline of its scope, then describe the parameters, then outputs, etc. The description of every function in the document must follow the same pattern. Such patterns include a lot of repeated words and repeated or similar word combinations, and some functions often share common functionality. As a result, we have a significant number of near duplicates; to be more precise, we have a hierarchy of such duplicates. These duplicates could be utilized for the formal reuse of text fragments.

The second attribute (format) is improved through polishing document language during duplicate management as picture captions, notes, phrases to introduce examples, author information, references to other documents, etc. – all these are made uni-

form. All these text fragments should be reused across the document. The unification of document markup features is also carried out (for example, using bold/italic typeface for terms), because these errors are easily observed during the analysis of duplicates.

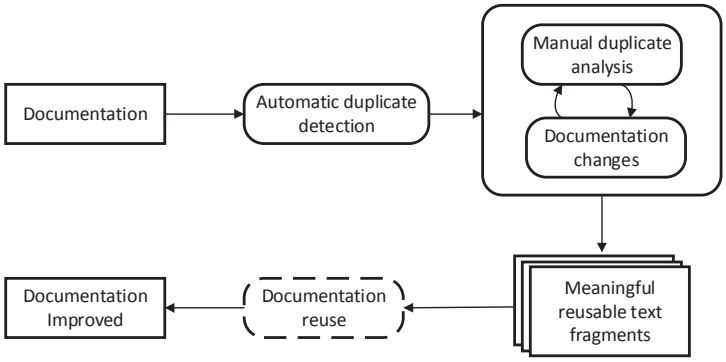

Fig. 1. Duplicate management process

The third attribute (spelling and grammar) is improved when the technical writer performs duplicate analysis and makes the document uniform. Across this activity a lot of spelling and grammer errors become obvious and could be corrected.

It is noteworthy that the automatic duplicate detection indicates the numeric documentation errors as variations in near duplicates. After these errors are corrected, the reuse amount of the document increases (reuse amount is defined in [20] as the relation of the length of all duplicates in a document to the document's size).

In fact, many other attributes mentioned in [19], such as accuracy, consistency (uniformity, integrity), correctness, readability, etc. are indirectly improved by the duplicate management process as well.

5 Example

We present the results of our pilot application of the duplicate management process for a user manual of a modelling toolkit. This toolkit provides enterprise architecture modeling facilities for a big corporation [21]. It includes diagram editors allowing to create and modify models of 10 types. By applying the duplicate management process, we found that the functionality of diagram creation in five editors is almost the same (only the names of model types and other small details were different). However, the corresponding text fragments in the user manual differed significantly. We uniformed these five text fragments achieving the average of 82% similarity for them. Similarity is calculated as follows. Firstly, we calculated four similarity values, where every value is the Levenshtein distance [22] between the first text fragment and one of the other four fragments. The first text fragment precedes the others in the document. Secondly, the average similarity is calculated. We classified documentation

changes to be made by means of quality attributes from [19]: information organization, format, spelling and grammar. Below these changes are described in detail.

- ***Information organization.*** When the document was being developed, the four text fragments were created by the copy-paste of the first one. Nevertheless, the structure of these text fragments was significantly changed by technical writers. Some paragraphs came in a different order; the pictures demonstrating duplicated functionality were presented in these text fragments in a chaotic manner (i.e. sometimes present, sometimes not). We corrected these problems, making a total of 25 document changes.
- ***Format.*** We uniformed the variations in describing the same actions and entities. For example, there were detected such variations as «After that push Next» and «After that push *button* Next». We made 7 document changes in total.
- ***Spelling and grammar.*** We corrected syntax errors and uniformed the use of bold/italic typeface for the terms. In total, we made 15 document changes.

6 Conclusions

In this paper, we have presented the duplicate management process for software reference documentation. In the future, we are going to further explore the process presented by focusing on its application to various types of reference documents. Another important research issue is providing the proper tool support of the process. In particular, documentation change should be integrated with duplicate management software. Also, reuse techniques should be developed and integrated both with duplicate management software and documentation development environments.

References

1. F.Brooks. The Mythical Man-Month. Addison-Wesley, Reading, Mass (1975).
2. I. Sommerville. Software Engineering, Vol 2: The Supporting Processes, chapter Software Documentation. Wiley-IEEE Press (2002).
3. D.Parnas. Precise Documentation: The Key to Better Software. The Future of Software Engineering: 125-148 (2010).
4. E.Juergens, et al. Can clone detection support quality assessments of requirements specifications? ICSE (2): 79-88 (2010).
5. A.Wingkvist, et. al. Analysis and visualization of information quality of technical documentation. ECIME: 388–396 (2010).
6. M. Oumaziz, et. al. Documentation Reuse: Hot or Not? An Empirical Study. ICSR 2017: 12-27 (2017).
7. D.Koznov, K. Romanovsky. DocLine: A method for software product lines documentation development. Programming and Computer Software 34(4): 216-224 (2008).
8. D. Koznov, Luciv D., et. al. Clone Detection in Reuse of Software Technical Documentation. LNCS, 9609: 170–185 (2016).
9. D. Luciv, D. Koznov, et. al. On fuzzy repetitions detection in documentation reuse. Programming and Comp. Soft., 42(4): 216–224 (2016).

10. D. Luciv, D. Koznov, et al. Detecting near duplicates in software documentation. Bulletin of the South Ural State University, Series: Math. Mod., Programming and Comp. Soft. Submitted (2017).
11. M.Horie, S.Chiba. Tool Support for Crosscutting Concerns of API Documentation. AOSD 2010: 97-108 (2010)
12. M.Nosal, J. Poruban. Reusable software documentation with phrase annotations. Central Europ. J. Computer Science 4(4): 242-258 (2014).
13. M.Nosal, J.Poruban. Preliminary report on empirical study of repeated fragments in internal documentation. FedCSIS: 1573-1576 (2016).
14. H.Basit, W.Smyth, et al. Efficient Token Based Clone Detection with Flexible Tokenization. In Proceedings of ACM SIGSOFT International Symposium on the Foundations of Software Engineering: 513–516 (2007).
15. K.Romanovsky, D.Koznov, L.Minchin. Refactoring the Documentation of Software Product Lines. LNCS, 4980: 158–170 (2011).
16. L.Kanteev, et. al. Discovering Near Duplicate Text in Software Documentation. Proceedings of ISP RAS. Accepted (2017).
17. J. Corbet. The present and future of formatted kernel documentation. https://lwn.net/Articles/671496/ (2016).
18. S. Wagner, D. Fernández. Analysing Text in Software Projects. https://arxiv.org/abs/1612.00164 (2016).
19. J.Zhi, et al. Cost, benefits and quality of software development documentation: A systematic mapping. Journal of Systems and Software, 99: 175-198 (2015).
20. W.Frakes, C.Terry. Software reuse: metrics and models. ACM Comput. Surv., 28(2): 415–435 (1996).
21. D. Koznov, et. al. Specifics of projects in the area of enterprise architecture development. Business Informatics, 4(34): 15-23 (2015).
22. V.Levenshtein. Binary codes with correction for deletions and insertions of the symbol 1. Problemy Peredachi Informacii 1(1): 12–25 (1965).