Discovering Near Duplicate Text in Software Documentation*

L.D. Kanteev <lkolt2@mail.ru>
Yu.O. Kostyukov <taxixx@inbox.ru>
D.V. Luciv <d.lutsiv@spbu.ru>
D.V. Koznov <d.koznov@spbu.ru>
M.N. Smirnov <m.n.smirnov@spbu.ru>
Saint Petersburg State University,
7/9 Universitetskaya emb., St. Petersburg, 199034, Russia

Abstract. Development of software documentation often involves copy-pasting, which produces a lot of duplicate text. Such duplicates make it difficult and expensive documentation maintenance, especially in case of long life cycle of software and its documentation. The situation is further complicated by duplicate information frequently being near duplicate, i.e., the same information may be presented many times with different levels of detail, in various contexts, etc. There are a number approaches to deal with duplicates in software documentation. But most of them use software clone detection technique, that is make difficult to provide efficient near duplicate detection: source code algorithms ignore a document structure, and they produce a lot of false positives. In this paper, we present an algorithm aiming to detect near duplicates in software documentation using natural language processing technique called as N-gram model. The algorithm has a considerable limitation: it only detects single sentences as near duplicates. But it is very simple and may be easily improved in future. It is implemented with use of Natural Language Toolkit (NLTK), and. Evaluation results are presented for five real life documents from various industrial projects. Manual analysis shows 39 % of false positives in automatic detected duplicates. The algorithm demonstrates reasonable performance: documents of 0,8–3 Mb are processed 5–22 min.

Keywords: software documentation, near duplicates, natural language processing, N-gram model.

DOI: 10.15514/ISPRAS-2017-29(4)-21

For citation: Kanteev L.D., Kostyukov Yu.O., Luciv D.V., Koznov D.V., Smirnov M.N. Discovering Near Duplicate Text in Software Documentation. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 303-314. DOI: 10.15514/ISPRAS-2017-29(4)-21

* This work is partially supported by RFBR grant No 16-01-00304

1. Introduction

Software projects produce a lot of textual information, and analysis of this data is a truly significant task for practice [1]. One particular problem in this context is software documentation duplicate management. When being developed, a lot of copy-pasted text fragments appeared in software documentation, which is often not tracked properly. According classification from [2], there are different kinds of software documents. For some of them, duplicate text is undesired, while others should contain duplicate text. But in any case duplicates increase documentation complexity and maintenance costs. The situation is further complicated by duplicate information frequently being “near duplicate”, i.e., the same information may be presented many times with different levels of detail, in various contexts, etc.

Most popular technique to detect duplicates in software documentation is software clone detection [3]. There are a number of approaches using this technique in software documentation research [4],[5],[6]. However, these approaches operate only with exact duplicates. Near duplicate clone detection techniques [7],[8],[9],[10] are not directly capable of detecting duplicates from text documents as they involve some degree of parsing of the underlying source code for duplicate detection.

In our previous studies [11],[12],[13] we have presented a near duplicate detection approach which is based on software clone detection. We adapted clone detection tool Clone Miner [14] to detect exact duplicates in documents, then near duplicates were extracted as combinations of exact duplicates. However, this approach outcomes a lot of false positives because it can not manage exact duplicate detection and operates with bad-quality “bricks” for combination of near duplicates. Meanwhile false positives’ problem is one of the big obstacle of duplicate management in practice [4].

In this paper we suggest an near duplicate detection algorithm based on N-gram model [1]. The algorithm doesn't use software clone detection, omitting the intermediate phases of exact duplicate detection. We have implemented the algorithm using Natural Language Toolkit [15] (NLTK). The algorithm was evaluated on documentation of five industrial projects.

2. Related Work

The problem of duplicate management in software project documents is being actively explored at the moment. Juergens et al. [4] analyze redundancy in requirement specifications. Horie et al. [16] consider the problem of text fragment duplicates in Java API documentation. Wingkvist et al. [5] detect exact duplicates to manage documents maintenance. Rago et al. [17] detect duplicate functionality in textual requirement specifications. However, the problem of near duplicate detection is still open. It is mentioned in [4], and Nosál and Porubän [18] suggest only using near duplicates omitting the way to detect them.

For software engineering, the conceptual background of near duplicate analysis is provided by Bassett [19]. He introduced the terms of archetype (the common part of various occurrences of variable information) and delta (the variation part). Based on this concept, Jarzabek developed an XML-based software reuse method [20]. Koznov

and Romanovsky [21],[22] applied the ideas of Bassett and Jarzabek to software documentation reuse, including automated documentation refactoring. However, these studies did not resolve the problem of document duplicate detection.

There are various techniques to detect near duplicate clones in source code. SourcererCC [7] detects near duplicates of code blocks using a static bag-of-tokens strategy that is resilient to minor differences between code blocks. Deckard [8] computes certain characteristic vectors of code to approximate the structure of Abstract Syntax Trees in the Euclidean space. Locality sensitive hashing (LSH) [9] is used to group similar vectors with the Euclidean distance. NICAD [10] is a text-based near duplicate detection tool that also uses a tree-based structural analysis. However, these techniques are not directly capable of detecting duplicates in text documents as they involve some degree of parsing the underlying source code for duplicate detection. A suitable customization for this purpose can be explored in the future.

Finally, there is a need for mature near duplicate detection methods to provide a proper duplicate analysis in software documentation. New information retrieving methods should be applied to increase the search quality. Natural language processing methods appear attractive for that purpose [1].

3. Background

Modern natural language processing and computer linguistics employ numerous standard approaches to analyze and transform texts. One of them is N-gram model [23]. Let us consider the text as a set of sentences. For every sentence the N-gram model includes all sequences (N-grams) consisting of n words, where every next word directly follow to previous one in the same order as in the sentence. Therefore every N-gram is a substring of the correspondent sentence. For example, if we want to detect the fact that two sentence are similar we can to compare their N-gram sets. N-gram model is used to perform different kinds of text analysis.

One of the most common programming tools for practical use of N-gram model is Natural Language Toolkit (NLTK) [15]. It provides a number of standard linguistic operations and is implemented in Python, that makes it easy to integrate NLTK into our Documentation Refactoring Toolkit [24] environment.

4. The Algorithm

The proposed algorithm requires the raw input document to be preprocessed: it should be divided into sentences, the sentences should be divided into words (tokens), and for every sentence an N-gram set is build. The algorithm collects document sentences into groups, if they are close to each other and were likely derived from one source by copy and paste.

The algorithm works as follows. First, it extracts sentences and builds 3-gram set for each of them. After that, for each sentence, the algorithm scans existing groups and chooses the best one, which already contains the largest number of the sentence's 3-grams. Then, if the best group already contains at least a half of the sentence's 3-grams, the sentence is added to this group, and the group's 3-gram set is

complemented with the new sentence's 3-grams. When no such group is found, a new group is introduced. Finally, the algorithm outputs the groups that contain two or more sentences. These groups are near duplicate groups.

```
1:  for i = 1 to size(sent) do
2:      curSent ← sent[i]
3:      bestOverlap ← 0
4:      bestGroup ← NULL
5:      for j = 1 to size(groups) do
6:          curGroup ← groups[j]
7:          curIntersect ← intersect(curSent.nGrams, curGroup.nGrams)
8:          curOverlap ← size(curIntersect) / size(curSent.nGrams)
9:          if curOverlap > bestOverlap then
10:             bestOverlap ← curOverlap
11:             bestGroup ← curGroup
12:         end if
13:     end for
14:     if bestOverlap < 0.5 then
15:         create new group newGroup
16:         newGroup.nGrams += curSent.nGrams
17:         newGroup.sent += curSent
18:     else
19:         bestGroup.nGrams += curSent.nGrams
20:         bestGroup.sent += curSent
21:     end if
22: end for
23: for all G in groups such that size(G) ← 1
24:     groups −= G
25: end for
26: return groups
```

27: *Algorithm 1. Specification of the algorithm*

Let's describe the algorithm in more detail. The formal specification of the algorithm is presented on the listing. Below the main functions of the algorithm are briefly considered.

- *intersect(A, B)* function returns elements, which exist in both *A* and *B* sets
- *size(A)* function returns number of elements in the set A
- *sent* is an array of sentences in document text
  - *sent[i].nGrams* is 3-gram set of the *i*-th sentence
- *groups* is an array of near duplicate groups
  - *groups[i].nGrams* is a 3-gram set of *i*-th group
  - *groups[i].sent* is a set of sentences of *i*-th group

Details of proposed algorithm are described below:

1. **Lines 1–22:** the main algorithm cycle, which iterates over all sentences of the document.
2. **Lines 5–13:** the cycle for the best group selection. For each groups:
   2.1. **Line 7:** intersection of 3-gram set with the 3-gram set of current sentence is calculated.
   2.2. **Line 8:** we calculate the ratio of this intersection size to total sentence 3-grams set size.
   2.3. **Lines 9–12:** if the current group is the best of processed ones, we remember it.
3. **Line 14:** we check if above ratio is less than 0.5, and:
   3.1. **Lines 15–17:** when it is less than 0.5, we create new group and put sentence into it.
   3.2. **Lines 19, 20:** otherwise, we put the sentence into the best group found.
4. **Lines 23–25:** groups with single sentence are not near duplicate groups, therefore we remove them.

5. Evaluation

We follow to the GQM framework [25] to organize evaluation of our algorithm. We formulate a set of evaluation questions:

Question 1: How many false positives (incorrect and irrelevant duplicate groups) and meaningful near duplicates are found?

Question 2: What is the performance of the algorithm?

We use the notion *reuse amount* [26] that means the relation of the reusable part to document length. For exact duplicates the reusable part is the total number of symbols, covered by duplicates, for near duplicates we consider only their archetypes. In [4] the same metric is named *clone coverage*.

Following [12], [13] we selected documentation of the four open sources as evaluation objects, but add one more commercial project documentation:

- Linux Kernel documentation (LKD), 892 KB in total [27];
- Zend Framework documentation (Zend), 2924 KB in total [28];
- DocBook 4 Definitive Guide (DocBook), 686 KB in total [29];
- Version Control with Subversion (SVN), 1810 KB in total [30];
- Commercial project user guide (CProj), 164 KB in total.

To answer **question 1**, we performed an manual analysis of near duplicate detected. The results are presented in Table 1.

The table includes column *Document* (evaluation documents) and two sections: *Proposed algorithm* (data concerning algorithm presented in the paper) and *Manual analysis* (results of manual analysis of the algorithm output). The *Proposed algorithm* section is organized as follows:

- *automatically detected* shows numbers of groups, which algorithm found;
- *raw reuse amount* contains reuse amount values for the evaluated documents.

The *Manual analysis* section contains the following columns:

- *markup-only* contains numbers of groups without human-readable text (they only contain markup);
- *irrelevant* presents numbers of false-positive groups, which were detected by human during manual revision of algorithm output;
- *total meaningful* shows number of meaningful duplicates, manually detected analyzing algorithm output;
- *meaningful reuse amount* presents reuse amount values for meaningful near duplicates detected.

Table 1. Near-duplicate groups detected

Document	Proposed algorithm						Manual analysis	
	Automatically detected	Raw reuse amount	Markup-only	Irrelevant	Total meaningful	Meaningful reuse amount	Total meaningful	Meaningful reuse amount
LKD	189	18.9%	20.1%	13.2%	66.7%	7.7%	15	5.1%
Zend	601	14.5%	10.3%	26.5%	63.2%	8.6%	27	2.1%
DocBook	73	13.0%	13.7%	32.9%	53.4%	3.2%	12	1.7%
SVN	349	10.2%	27.8%	21.5%	50.7%	5.0%	16	2.3%
CProj	72	38.3%	0.0%	29.2%	70.8%	29.5%	9	14.1%
Average		19.0%	14.4%	24.6%	61.0%	10.8%		5.0%

14.4% of groups contain no human-readable text, but only markup, 24.6% of groups contain text which is similar, but this is just formal similarity, and duplicates of those groups are not semantically connected. Remaining 61% of groups are meaningful duplicate groups. For documents of different sizes their count varies from few dozens to several hundreds depending on the size and nature of document, therefore we can say that proposed algorithm detects considerable amount of near duplicates, and most of them are meaningful. The reuse amount has been decreased in 2 times after manual processing. These data indicates the false positive problem need to be resolved for the algorithm.

Finally, to answer **question 2** we estimated the working time of the algorithm with the evaluation documents. For our experiments we used the usual work station Intel

i5-2400, 3.10GHz, RAM 4 GiB, Windows 10. Our estimation results are presented in table 2. The first column of the table contains the acronyms of the documents to be evaluated. The second one contains the size of the documents. The third column presents the algorithm processing time values. The forth column presents the processing speed. The processing speed depends on two parameters: the size of the document and the reuse amount. It decreases when the document size grows and as the reuse amount increases. The first statement is obvious. The second one follows from the fact that, roughly speaking, the larger the reuse amount is, the fewer groups of single sentence exist, and therefore number operations in cycle of the best group selection (see listing 1, lines 5-13) decreases. However, this is a rough estimation because the size of the groups also contributes to the processing speed. And we cannot say for certain whether or not a larger reuse amount might compensate for a larger document size. Among the five documents presented in table 2, we can see our assumption confirmed. In the case of these documents, the processing speed decreases as the document size increases, with one exception. The processing speed of the algorithm for Zend was higher than that for SVN, although the size of the Zend document was bigger than that of SVN. At the same time, the reuse amount of Zend is substantially higher than that of SVN. Also the assumption concerning the reuse amount works well in our experiments carried out outside of results presented in this paper. However, further research is needed to verify this assumption. In addition, implementation factors need to be explored, which can influence the algorithm performance. Finally, the performance of the algorithm appears sufficient for practical applications. The algorithm demonstrates an acceptable processing time for rather large documents, i.e. from 1 to 3 Mb. Larger documents are quite rare in practice.

Table 2. Performance analysis

Document	Size, Kb	Processing time, min	Processing speed, Kb/min
LKD	892	5.30	168.35
Zend	2924	22.14	132.08
DocBook	686	2.02	339.60
SVN	1810	17.14	105.59
CProj	164	0.17	946.15

6. Conclusion

We have presented an algorithm for the detection of near duplicates in software documentation based on N-gram model. The proposed algorithm is close to the naive voting clustering algorithm [31], using a similarity measure resembling the Jaccard index [32]. Compared to [12],[13], the algorithm looks much simpler, while also making use of the techniques and apparatus conventionally used for text analysis. It should be noted, the algorithm has a considerable limitation: it only detects single sentences as near duplicates. Our primary goal for future research is to extend the

algorithm to make possible processing arbitrary text fragments. Here are some additional future directions of the research:

1. It is necessary to resolve false positives problem. The algorithm output should be compared to manual document analysis.
2. Classification of false positives and meaningful near duplicates should be developed. False positives may include markup, document metadata, etc. Meaningful near duplicates usually describe entities of the same nature (function descriptions, command line parameters, data type specifications, etc.).
3. Improvement of experiment model should be performed. For example, Juergens et al. [4] spend much effort to obtain objective results in analyzing duplicates of real industry documents.

Research results could be applied in various fields of software engineering, e.g. in model based testing [33],[34] to provide correctness of initial requirement specifications, which are used for test generation.

References

[1] Wagner S., Fernández D.M. Analysing Text in Software Projects. Preprint, 2016. URL: https://arxiv.org/abs/1612.00164

[2] Parnas D. L. Precise Documentation: The Key To Better Software. Nanz S. (ed.) The Future of Software Engineering, Springer, 2011. DOI: 10.1007/978-3-642-15187-3_8

[3] Akhin, M., Itsykson, V. Clone Detection: Why, What and How? Proceedings of CEE-SECR'10, 2010, pp. 36–42. DOI: 10.1109/CEE-SECR.2010.5783148

[4] Juergens E. et al. Can clone detection support quality assessments of requirements specifications? Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering, 2010, vol. 2, pp. 79–88.

[5] Wingkvist A., Ericsson M., Lincke R., Löwe W. A Metrics-Based Approach to Technical Documentation Quality. Proceedings of 7th International Conference on the Quality of Information and Communications Technology, 2010, pp. 476–481.

[6] Nosál M., Porubän J. Preliminary report on empirical study of repeated fragments in internal documentation. Proceedings of the Federated Conference on Computer Science and Information Systems, Gdansk, 2016, pp. 1573–1576.

[7] Sajnani H., Saini V., Svajlenko J., Roy C.K., Lopes C.V. Sourcerercc: Scaling code clone detection to big-code. Proceedings of the 38th International Conference on Software Engineering, ACM, New York, USA, 2016, pp. 1157–1168. DOI: 10.1145/2884781.2884877

[8] Jiang L., Misherghi G., Su Z., Glondu S. DECKARD: Scalable and accurate tree-based detection of code clones. Proceedings of 29th International Conference on Software Engineering. Institute of Electrical and Electronics Engineers, 2007, pp. 96–105. DOI: 10.1109/ICSE.2007.30

[9] Huang T.K., Rahman M.S., Madhyastha H.V., Faloutsos M., Ribeiro B. An analysis of socware cascades in online social networks. Proceedings of the 22Nd International Conference on World Wide Web, 2013, pp. 619–630.

[10] Cordy J.R., Roy C.K.: The NiCad clone detector. Proceedings of the 19th IEEE International Conference on Program Comprehension. Institute of Electrical and Electronics Engineers, 2011, pp. 219–220. DOI: 10.1109/ICPC.2011.26

[11] Lutsiv D.V., Koznov D.V., Basit H.A., Lieh O.E., Smirnov M.N., Romanovsky K.Yu. An approach for clone detection in documentation reuse. Nauchno-tehnicheskij vestnik informacionnyh tehnologij, mehaniki i optiki [Scientific and Technical Journal of Information Technologies, Mechanics and Optics] vol. 92, issue 4, 2014, pp. 106–114 (in Russian).
[12] Koznov D. et al. Clone detection in reuse of software technical documentation. Mazzara M., Voronkov A. (eds.), International Andrei Ershov Memorial Conference on Perspectives of System Informatics, 2015; Lecture Notes in Computer Science, vol. 9609, 2016, pp. 170–185. DOI: 10.1007/978-3-319-41579-6_14
[13] Luciv D., Koznov D., Basit H.A., Terekhov A.N. On fuzzy repetitions detection in documentation reuse. Programming and Computer Software, vol. 42, issue 4, 2016, pp. 216–224. DOI: 10.1134/s0361768816040046
[14] Basit H.A., Smyth W.F., Puglisi S.J., Turpin A., Jarzabek S. Efficient Token Based Clone Detection with Flexible Tokenization. Proceedings of ACM SIGSOFT International Symposium on the Foundations of Software Engineering, ACM Press, 2007, pp. 513–516. DOI: 10.1145/1295014.1295029
[15] Natural Language Toolkit, URL: http://nltk.org/
[16] Horie M., Chiba S. Tool support for crosscutting concerns of API documentation. Proceedings of 9th International Conference on Aspect-Oriented Software Development, 2010, pp. 97–108. DOI: 10.1145/1739230.1739242
[17] Rago A., Marcos C., Diaz-Pace J.A. Identifying duplicate functionality in textual use cases by aligning semantic actions. International Journal on Software and Systems Modeling, vol. 15, issue 2, 2016, pp. 579–603. DOI: 10.1007/s10270-014-0431-3
[18] Nosál' M., Porubän J. Reusable software documentation with phrase annotations. Open Computer Science, vol. 4, issue 4, 2014, pp. 242-258. DOI: 10.2478/s13537-014-0208-3
[19] Bassett P. Framing software reuse – lessons from real world. Prentice Hall, 1996. ISBN: 0-13-327859-X
[20] Jarzabek S., Bassett P., Zhang H., Zhang W. XVCL: XML-based Variant Configuration Language. Proceedings of 25th International Conference on Software Engineering, 2003, pp. 810–811. DOI: 10.1109/ICSE.2003.1201298
[21] Koznov D., Romanovsky K.. DocLine: A Method for Software Product Lines Documentation Development. Programming and Computer Software, vol. 34, issue 4, 2008, pp. 216–224. DOI: 10.1134/S0361768808040051
[22] Romanovsky K., Koznov D., Minchin L. Refactoring the Documentation of Software Product Lines. Central and East European Conference on Software Engineering Techniques, Brno (Czech Republic), 2008; Lecture Notes in Computer Science, vol. 4980, Springer, 2011, pp. 158–170. DOI: 10.1007/978-3-642-22386-0_12
[23] Broder A.Z. et al. Syntactic clustering of the web. Computer Networks and ISDN Systems. vol. 29, issue 8, 1997, pp. 1157–1166. DOI: 10.1016/S0169-7552(97)00031-7
[24] Documentation Refactoring Toolkit, URL: http://www.math.spbu.ru/user/kromanovsky/docline/index_en.html
[25] Basili V., Caldiera G., Rombach H. The Goal Question Metric Approach. Encyclopedia of Software Engineering, Wiley, 1994. DOI: 10.1002/0471028959.sof142
[26] Frakes W., Terry C.. Software reuse: metrics and models. ACM Computing Surveys, vol. 28, issue 2, 1996, pp. 415–435. DOI: 10.1145/234528.234531
[27] Linux Kernel Documentation, snapshot on Dec 11, 2013. URL: https://github.com/torvalds/linux/tree/master/Documentation/DocBook/
[28] Zend PHP Framework documentation, snapshot on Apr 24, 2015. URL: https://github.com/zendframework/zf1/tree/master/documentation

[29] DocBook Definitive Guide, snapshot on Apr 24, 2015. URL: http://sourceforge.net/p/docbook/code/HEAD/tree/trunk/defguide/en/
[30] SVN Book, snapshot on Apr 24, 2015. URL: http://sourceforge.net/p/svnbook/source/HEAD/tree/trunk/en/book/
[31] Braun R.K., Kaneshiro R. Exploiting topic pragmatics for new event detection. Technical report. National Institute of Standards and Technology, Topic Detection and Tracking Workshop, 2004.
[32] Jaccard P. Distribution de la flore alpine dans le Bassin des Dranses et dans quelques regions voisines [Distribution of Alpine flora in the Dranses Basin and some neighboring regions]. Bulletin de la Société Vaudoise des Sciences Naturelles [Bulletin of the Vaudois Society of Natural Sciences], vol. 140, issue 37, 1901, pp. 241–272 (in French)
[33] Drobintsev P.D., Kotlyarov V. P., Letichevsky A.A. A formal approach to test scenarios generation based on guides. Automatic Control and Computer Sciences, vol. 48, issue 7, 2014, pp. 415–423. DOI: 10.3103/S0146411614070062
[34] Zelenov S.V., Silakov D.V., Petrenko A.K., Conrad M., Fey I. Automatic test generation for model-based code generators. Proceedings of 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, pp. 75–81. DOI: 10.1109/ISoLA.2006.70

Обнаружение неточно повторяющегося текста в документации программного обеспечения *

Л.Д. Кантеев <lkolt2@mail.ru>
Ю.О. Костюков <taxixx@inbox.ru>
Д.В. Луцив <d.lutsiv@spbu.ru>
Д.В. Кознов <d.koznov@spbu.ru>
М.Н. Смирнов <m.n.smirnov@spbu.ru>
Санкт-Петербургский государственный университет,
199034, Россия, Санкт-Петербург, Университетская набережная 7/9

Аннотация. При создании документации программного обеспечения часто применяется копирование и вставка с последующим редактированием, в результате чего возникает много повторяющегося текста. Такие повторы усложняют и удорожают поддержку документации, особенно в случае длительных жизненных циклов программного обеспечения и документации. Ещё более усложняет ситуацию то, что зачастую информация повторяется приблизительно, т.е. одна и та же информация может быть многократно представлена с разными уровнями детализации, в различных контекстах и т.д. В данной работе предложен алгоритм, предназначенный для обнаружения неточных повторов в документации программного обеспечения. Алгоритм основан на модели N-грамм и реализован с использованием Natural Language Toolkit. Алгоритм апробирован на документации нескольких проектов с открытым исходным кодом.

Ключевые слова: документация программного обеспечения, нечёткие повторы, обработка текстов на естественных языках, модель N-грамм.

DOI: 10.15514/ISPRAS-2017-29(4)-21

* Работа частично поддержана грантом РФФИ №16-01-00304

Для цитирования: Кантеев Л.Д., Костюков Ю.О., Луцив Д.В., Кознов Д.В., Смирнов М.Н. Обнаружение неточно повторяющегося текста в документации программного обеспечения. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 303-314 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-21

Список литературы

[1] Wagner S., Fernández D.M. Analysing Text in Software Projects. Preprint, 2016. URL: https://arxiv.org/abs/1612.00164

[2] Parnas D. L. Precise Documentation: The Key To Better Software. Nanz S. (ed.) The Future of Software Engineering, Springer, 2011. DOI: 10.1007/978-3-642-15187-3_8

[3] Akhin, M., Itsykson, V. Clone Detection: Why, What and How? Proceedings of CEE-SECR'10, 2010, pp. 36–42. DOI: 10.1109/CEE-SECR.2010.5783148

[4] Juergens E. et al. Can clone detection support quality assessments of requirements specifications? Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering, 2010, vol. 2, pp. 79–88.

[5] Wingkvist A., Ericsson M., Lincke R., Löwe W. A Metrics-Based Approach to Technical Documentation Quality. Proceedings of 7th International Conference on the Quality of Information and Communications Technology, 2010, pp. 476–481.

[6] Nosál M., Porubän J. Preliminary report on empirical study of repeated fragments in internal documentation. Proceedings of the Federated Conference on Computer Science and Information Systems, Gdansk, 2016, pp. 1573–1576.

[7] Sajnani H., Saini V., Svajlenko J., Roy C.K., Lopes C.V. Sourcerercc: Scaling code clone detection to big-code. Proceedings of the 38th International Conference on Software Engineering, ACM, New York, USA, 2016, pp. 1157–1168. DOI: 10.1145/2884781.2884877

[8] Jiang L., Misherghi G., Su Z., Glondu S. DECKARD: Scalable and accurate tree-based detection of code clones. Proceedings of 29th International Conference on Software Engineering. Institute of Electrical and Electronics Engineers, 2007, pp. 96–105. DOI: 10.1109/ICSE.2007.30

[9] Huang T.K., Rahman M.S., Madhyastha H.V., Faloutsos M., Ribeiro B. An analysis of socware cascades in online social networks. Proceedings of the 22Nd International Conference on World Wide Web, 2013, pp. 619–630.

[10] Cordy J.R., Roy C.K.: The NiCad clone detector. Proceedings of the 19th IEEE International Conference on Program Comprehension. Institute of Electrical and Electronics Engineers, 2011, pp. 219–220. DOI: 10.1109/ICPC.2011.26

[11] Луцив Д.В., Кознов Д.В., Басит Х.А., Ли О.Е., Смирнов М.Н., Романовский К.Ю. Метод поиска повторяющихся фрагментов текста в технической документации. Научно-технический вестник информационных технологий, механики и оптики, т. 92, вып. 4, 2014, стр. 106–114.

[12] Koznov D. et al. Clone detection in reuse of software technical documentation. Mazzara M., Voronkov A. (eds.), International Andrei Ershov Memorial Conference on Perspectives of System Informatics, 2015; Lecture Notes in Computer Science, vol. 9609, 2016, pp. 170–185. DOI: 10.1007/978-3-319-41579-6_14

[13] Луцив Д.В., Кознов Д.В., Басит Х.А., Терехов А.Н. Задача поиска нечётких повторов при организации повторного использования документации. Программирование, т. 42, № 4, 2016, стр. 39–49.

[14] Basit H.A., Smyth W.F., Puglisi S.J., Turpin A., Jarzabek S. Efficient Token Based Clone Detection with Flexible Tokenization. Proceedings of ACM SIGSOFT International Symposium on the Foundations of Software Engineering, ACM Press, 2007, pp. 513–516. DOI: 10.1145/1295014.1295029

[15] Natural Language Toolkit, URL: http://nltk.org/

[16] Horie M., Chiba S. Tool support for crosscutting concerns of API documentation. Proceedings of 9th International Conference on Aspect-Oriented Software Development, 2010, pp. 97–108. DOI: 10.1145/1739230.1739242

[17] Rago A., Marcos C., Diaz-Pace J.A. Identifying duplicate functionality in textual use cases by aligning semantic actions. International Journal on Software and Systems Modeling, vol. 15, issue 2, 2016, pp. 579–603. DOI: 10.1007/s10270-014-0431-3

[18] Nosál' M., Porubän J. Reusable software documentation with phrase annotations. Open Computer Science, vol. 4, issue 4, 2014, pp. 242-258. DOI: 10.2478/s13537-014-0208-3

[19] Bassett P. Framing software reuse – lessons from real world. Prentice Hall, 1996. ISBN: 0-13-327859-X

[20] Jarzabek S., Bassett P., Zhang H., Zhang W. XVCL: XML-based Variant Configuration Language. Proceedings of 25th International Conference on Software Engineering, 2003, pp. 810–811. DOI: 10.1109/ICSE.2003.1201298

[21] Кознов Д.В., Романовский К.Ю. DocLine: метод разработки документации семейства программных продуктов. Программирование, т. 34, вып. 4, 2008, С. 1–13.

[22] Romanovsky K., Koznov D., Minchin L. Refactoring the Documentation of Software Product Lines. Central and East European Conference on Software Engineering Techniques, Brno (Czech Republic), 2008; Lecture Notes in Computer Science, vol. 4980, Springer, 2011, pp. 158–170. DOI: 10.1007/978-3-642-22386-0_12

[23] Broder A.Z. et al. Syntactic clustering of the web. Computer Networks and ISDN Systems. vol. 29, issue 8, 1997, pp. 1157–1166. DOI: 10.1016/S0169-7552(97)00031-7

[24] Documentation Refactoring Toolkit, URL: http://www.math.spbu.ru/user/kromanovsky/docline/index.html

[25] Basili V., Caldiera G., Rombach H. The Goal Question Metric Approach. Encyclopedia of Software Engineering, Wiley, 1994. DOI: 10.1002/0471028959.sof142

[26] Frakes W., Terry C.. Software reuse: metrics and models. ACM Computing Surveys, vol. 28, issue 2, 1996, pp. 415–435. DOI: 10.1145/234528.234531

[27] Linux Kernel Documentation, snapshot on Dec 11, 2013. URL: https://github.com/torvalds/linux/tree/master/Documentation/DocBook/

[28] Zend PHP Framework documentation, snapshot on Apr 24, 2015. URL: https://github.com/zendframework/zf1/tree/master/documentation

[29] DocBook Definitive Guide, snapshot on Apr 24, 2015. URL: http://sourceforge.net/p/docbook/code/HEAD/tree/trunk/defguide/en/

[30] SVN Book, snapshot on Apr 24, 2015. URL: http://sourceforge.net/p/svnbook/source/HEAD/tree/trunk/en/book/

[31] Braun R.K., Kaneshiro R. Exploiting topic pragmatics for new event detection. Technical report. National Institute of Standards and Technology, Topic Detection and Tracking Workshop, 2004.

[32] Jaccard P. Distribution de la flore alpine dans le Bassin des Dranses et dans quelques regions voisines. Bulletin de la Société Vaudoise des Sciences Naturelles, vol. 140, issue 37, 1901, pp. 241–272 (франц.)

[33] Drobintsev P.D., Kotlyarov V. P., Letichevsky A.A. A formal approach to test scenarios generation based on guides. Automatic Control and Computer Sciences, vol. 48, issue 7, 2014, pp. 415–423. DOI: 10.3103/S0146411614070062

[34] Zelenov S.V., Silakov D.V., Petrenko A.K., Conrad M., Fey I. Automatic test generation for model-based code generators. Proceedings of 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, pp. 75–81. DOI: 10.1109/ISoLA.2006.70