A Topic Modeling Approach To Evaluate The Comments Consistency To Source Code

Martina Iammarino
Dep. of Engineering
University of Sannio
Benevento, Italy
iammarino@unisannio.it

Lerina Aversano
Dep. of Engineering
University of Sannio
Benevento, Italy
aversano@unisannio.it

Mario Luca Bernardi
Dep. of Engineering
University of Sannio
Benevento, Italy
bernardi@unisannio.it

Marta Cimitile
Unitelma Sapienza University
Rome, Italy
marta.cimitile@ unitelmasapienza.it

***Abstract* – A significant amount of source code in software systems is made up of comments, parts of the code that are ignored by the compiler. Comments in the code are a primary source for system documentation. These are crucial for the work of software maintainers, as a basis for code traceability, for maintenance activities, but also for the use of the code itself as a library or framework in other projects. Although many software developers consider comments important, existing approaches to software quality analysis mainly disregard code comments and focus only on source code. This paper presents an approach, based on topic modeling, for analyzing the comments consistency to the source code. A model was provided to analyze the quality of comments in terms of consistency since comments should be consistent with the source code they refer to. The results show a similarity in the trend of topic distribution and it emerges that almost all classes are associated with no more than 3 topics.**

***Keywords* –comment, topic modeling, Natural Language Toolkit.**

I. Introduction

The comments in the code represent the main source of documentation for a software system and are therefore fundamental for understanding the source code itself, both during the development phase and during the maintenance phase [9]. Everybody who is writing software knows the value of good comments [16].

As early as the early 1980s, Elshoff and Marcotty said that the comments help to understand the program and therefore reduce maintenance costs [19]. This discovery has also been confirmed by Ted Tenny's studies [12]. But as Lakhotia's example shows, sometimes programmers don't care about comments, not thinking that anyone else wants to understand the source code [20]. Although developers commonly agree on the importance of software documentation, comments are often overlooked due to release deadlines and other time pressures during development [13].

Nevertheless, fundamental for greater readability of the code [17][18], comments are considered the second documentary artifact most used to understand the code, behind only the code itself, because of express programmers' intentions in a more explicit manner [10]. Developers agree that poor general documentation often leads to misunderstandings, and several studies have shown that poor documentation significantly reduces the maintainability of the software [2].

For this reason, high-quality comments are crucial: a well-commented source code makes it easier for software designers to grasp general design decisions and understand the details of the implementation to make change requests or corrections of bugs [12].

Another important aspect has been addressed by Fluri et al. [2] which invalidated the fear that comments may be inconsistent. In their work, the authors show that in 90% of the cases in their study, changes to the source code simultaneously cause an adaptation of the corresponding comment. As a result, comments are usually updated with the evolving code.

Recent approaches, currently used for quality analysis ignore them or limit themselves to the calculation of their density in the system, thus stopping at a quantitative analysis [14][15].

Many previous studies make a quantitative assertion about the relationship presence of comments in the system, counting the number of lines containing a comment, divided by the total number of lines of code in the system [2], [3]. This metric, however, is too simple to be meaningful for several reasons. Ignore that some comments, such as copyright or commented code, do not promote understanding of the system or improve the quality of the system documentation.

There are three main reasons why the quality of comments has so far been overlooked. The first is that each comment has the same syntactic scheme, regardless of its purpose. The second is that the quality of the comments was defined only by the expectations of the software developers. In other words, a comment is considered high quality if it helps understand the code. However, beyond the subjective expectations of the developers, there is no precise definition of the quality of the comments. The third reason is that there are currently no automatic or semi-automatic methods for evaluating the quality of comments since their analysis is not very mandatory and is written in natural language [21].

The proposed study, on the other hand, aims to evaluate the quality of comments in terms of consistency with the source code, focusing exclusively on its content. In this work, we will present an approach, based on the Topic Modeling technique, and Natural Language Processing. The topic modeling refers to statistical models used to extract topics that occur in a collection of documents. A single topic is a set of similar words, each one associated to weight expressing the probability that it is related to the topic. Therefore, this technique assumes that a document is generally composed of multiple topics in different proportions.

The rest of the paper is structured as follows: in the following section related works are discussed; some background information is provided in section III; in section IV, the proposed approach is described, while the experiment results are discussed in Section V; finally, in Section VI and VII respectively, the threats to validity and the conclusions are reported.

II. RELATED WORK

This section contains similar works that have been conducted over the years on the comment topic modeling.

G. Antoniol et al. [1] propose a method based on information retrieval to retrieve traceability links between source code and free-text documents. They applied both a probabilistic model and a vector space information retrieval model in two case studies to trace the C ++ source code on manual pages and the Java code on functional requirements.

Beat Fluri et al. [2] describe an approach to map code and comments to observe coevolution in multiple versions. They studied three open-source systems and described how comments and code evolved over time. Their results show that: i) the newly added code - despite its growth rate - is just commented; ii) class and method statements are commented more frequently but much less; iii) 97% of changes to comments are made in the same revision as the modification of the associated source code.

Tan et al. [5] Using Natural Language Processing automatically analyzes the comments written in natural language to extract implicit rules of the program and use these rules to automatically detect inconsistencies between comments and source code, indicating bugs or incorrect comments.
Although their method has achieved high accuracy in detecting topic-specific comments, it cannot be applied to arbitrary comments.

Hazeline U. Asuncion et al. [3] have developed an automated technique that combines traceability with a machine learning technique known as argument modeling. Their approach automatically records traceability links during the software development process and learns a probabilistic thematic model of artifacts. The learned model allows the semantic categorization of artifacts and the topical display of the software system.

Wei Emma Zhang et al. [4] examine the problem of mining topics (i.e., topic extraction) from source code, which can facilitate the comprehension of the software systems. They propose a topic extraction method, Embedded Topic Extraction (EmbTE), that considers word semantics, which is never considered in mining topics from source code, by leveraging word embedding techniques. They also adopt Latent Dirichlet Allocation (LDA) and Non-negative Matrix Factorization (NMF) to extract topics from source code. Moreover, an automated term selection algorithm is proposed to identify the most contributory terms from source code for the topic extraction task.

McBurney and McMillan presented a novel approach for automatically generating summaries of Java methods that summarize the context surrounding a method [7] and also [6] conducted an empirical study examining method comments of source code written by authors, readers, and automatic source code summarization tools. Their work discovered that the accuracy of a human-written method comment could be estimated by the textual similarity of that method comment to the source code, addressing that a good comment written by developers should have a high semantic similarity to source code. Malik et al. (2008)

Steidl et. Al [21] provided a model for the quality of comments which is based on different categories of comments. To classify the comments, they used machine learning on Java and C / C ++ programs. The model includes several quality aspects. The validity of the metrics is assessed with a survey of 16 experienced software developers.

Chen 's et al. [8] method, based on machine learning, used code snippet and comment functionality to automatically detect the scope of comments on source code in Java programs. This method achieved a high accuracy of 81.45% by providing a solution for the mapping of the code-comments. It improved the performance of the base methods in both activities, which showed that the method is conducive to the automatic analysis and mining approaches to software repositories.

This study focuses on extracting topics from source code and comments, with the aims of evaluating the quality of comments in terms of consistency with the source code. More specifically, the goal is to identify the distribution of the topics extracted from the comments text respect to the source code classes of the software projects.

III. EXTRACTING TOPICS WITH LATENT DIRICHLET ALLOCATION

Latent Dirichlet Allocation is the method used in this work for modeling topics, a process of identifying topics in a series of documents.

The Latent Dirichlet Allocation model is an unsupervised text mining technique, it is widely applied for topic modeling, and successfully it extracts topics from texts written in natural language. It was developed by David Blei and is based on a Bayesian probabilistic approach [22].

It assumes that there are k topics across all of the documents, distribute these k topics across document m by assigning each word a topic.

For each word w in document m, it assumes its topic is wrong, but every other word is assigned the correct topic.

Probabilistically it assigns word w a topic based on two things: what topics are in the document, and how many times the word w has been assigned to a particular topic across all of the documents.

Finally, repeat this process a number of times for each document.

The Figure 1 shows what is known as a plate diagram of an LDA model where: α is the per-document topic distributions, β is the per-topic word distribution, θ is the topic distribution for document *m, φ* is the word distribution for topic *k, z* is the topic for the *n*-th word in document *m*, and *w* is the specific word.

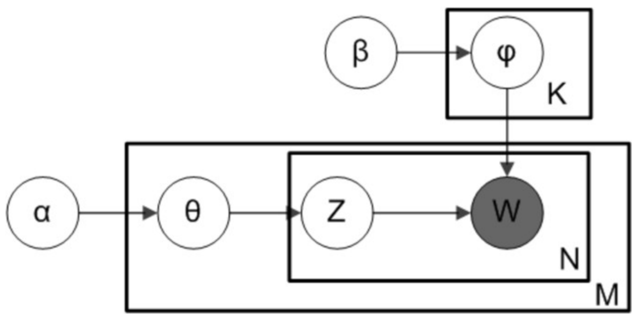


Fig 1: LDA Model

Kullback Leibler Divergence. The Kullback-Leibler divergence is a measure that finds the difference between two probability distributions.

The divergence KL, which is closely related to relative entropy, information divergence, and information for discrimination, is a non-symmetric measure of the difference between two probability distributions p (x) and q(x).

In particular, the Kullback-Leibler (KL) divergence of q (x) from p (x), indicated by DKL (p (x), q (x)), is a measure of the information lost when using q (x) for approximate p (x). Let p (x) and q (x) two probability distributions of a discrete random variable x. That is, both p (x) and q (x) are added to 1, ep (x)> 0 and q (x)> 0 for any x in X. DKL (p (x), q (x)) is defined as follows:

$$D_{KL}(P \parallel Q) = \sum_i P(i) \log_2 \frac{P(i)}{Q(i)}$$

The KL divergence measures the expected number of extra bits required to encode samples from p (x) when using a code based on q (x), rather than using a code based on p (x). Typically, p (x) represents the "true" distribution of data, observations or a theoretical distribution calculated with precision. The measure q (x) generally represents a theory, a model, a description or an approximation of p (x).

In the simple case, a Kullback-Leibler divergence of 0 indicates that the two distributions in question are identical.

In this study, it is used between document topic distributions to find the most likely similar and relevant topics from the two corpora: source code and comments.

IV. Approach

In this section, we describe carefully the approach used to extract and analyze the topics distributions. Figure 2 shows in detail the process used, highlighting the main stages. Specifically, all the comments and source code in the software classes were extracted for each of the analyzed projects. These, after a pre-processing phase, constitute the two corpora, one relating to comments and one relating to source code. These allowed the extraction of topics, developing the LDA model. Once the arguments of both corpora were generated, the divergence was calculated. Finally, the results obtained were analyzed. More specifically, all the steps are explained in detail.

Comment text and source code extraction

The input of this phase is the Java project to be analyzed. For the extraction of comments and source code from files, the regular expressions were used.
The following regular expressions have been used:

"//.*\n" for comments on one line;
"/*+.*?*/" for comments on several lines.

In particular, in the case of the comments, they were used to filter the strings that coincide with the model, while in the case of the source code they were used to filter the strings that do not correspond to the model.
The outputs of this phase are two files, one containing the comment's text and the other containing the source code of each Java project analyzed.

.

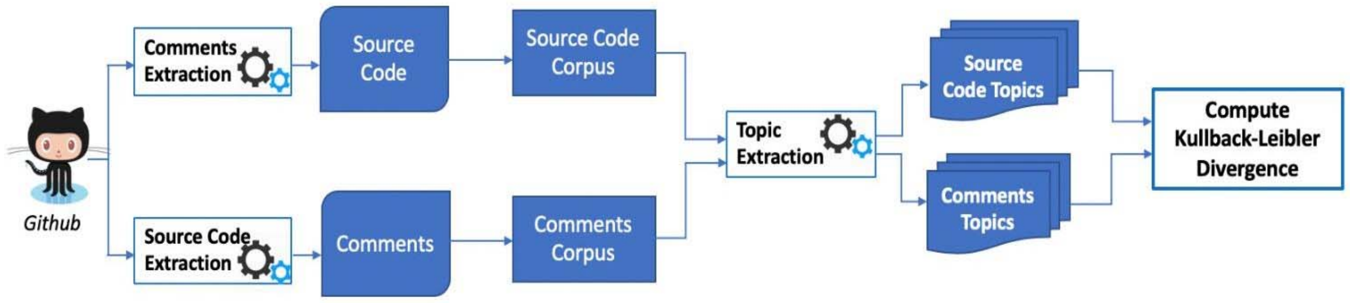


Fig. 2 : Overview of the tools chain and steps of the study

Corpus preprocessing. Before proceeding with the modeling of the texts, a phase of preparation and cleaning of the data is necessary. The most relevant operations of this process are the generation and removal of uncommon stopwords and symbols. It is a very delicate phase since it is possible to reduce two words with completely different meanings to the same root and it is, therefore, necessary to carefully choose the algorithm to be used. The second operation concerns the elimination of passwords, which is the series of terms commonly used in the language and present in all texts, such as articles and conjunctions since these do not add any information on the subject. The punctuation symbols have been replaced with white spaces and tokenization of the text has been carried out.

Topics extraction. Corpora related to comments and source code were used to execute and train the LDA argument extraction model. To do this, the Gensim library[1] wasused.

Kullback-Leibler divergence computation between the documents of Source Code and Comments. The K-L approach has been used to extract relevance among the linguistic topics of the corpus. A score called consistency score has been associated with each topic: the higher it is, the more words that form the topic are semantically related to each other.

Results Analysis. Two investigations were conducted to compare the results obtained:

RQ1 What is the comments topics distribution on source code and comments files?
RQ2 What is the comments topics incidence per source code files and comment files?

V. EXPERIMENTAL RESULTS

In this section, we describe the experimental setup and the datasets used in our experiments, followed by the evaluation of the proposed approach. We conducted our experiments over 4 open-source systems. We performed LDA topics modelling for both of their source code system and over their source code comments. The evaluation of the new approach is done by comparing how many relevant topics from both corpora were retrieved as relevant in the retrieved list, and the number of traceability links that exist between the two corpora.

Two different experiments were conducted: i) the first with the aim of tracing the comments topics on the source code topics, to understand if there is any consistency between them; ii) the second, to trace the comments topics on the source code comments file, to understand the distribution of the extracted over the different files composing the software project.

The context of the study consists of the evolution of four open Java source projects: Jedit, Jabref, JFreeChart, and Spring, for which Table 1 reports the release considered and the classes' number. In all cases, a topic number was set.

Project	Release	#Classes
Jedit	4.5	554
Jabref	2.7.2	598
JFreeChart	1.0.19	1017
Spring	5.1.3	6922

Table 1: Software projects analyzed

RQ1 What is the comments topics distribution on source code and comments files.

In this analysis, the presence of each previously extracted comments topics was traced within the source code files and comments files.

To support this investigation, for each project, a bar chart was produced, which reports the number of classes in which these topics were recognized.

Specifically, Figure 3 shows with gray bars the number of classes where the topics were found in the source code files, while blue bars represent the number of classes where the topics were retrieved in the comment's files.

Overall, it emerges that all the topics are almost equally traced in the source code and the comment text.

In particular, in Jedit, topics 12, 14 and 21 are traced with a large number of source code classes and comments files, while the other topics are similarly distributed.

Furthermore, it is important to underline that topic 9 and 21 are traced to the source code classes significantly more than to the comment files. This means that these topics are more represented in the code rather than in comments files.

In Jabref, the results are similar to those just illustrated for Jedit. Again, there are two main topics, 0 and 24 mainly retrieved in the source code rather than in the comment's files.

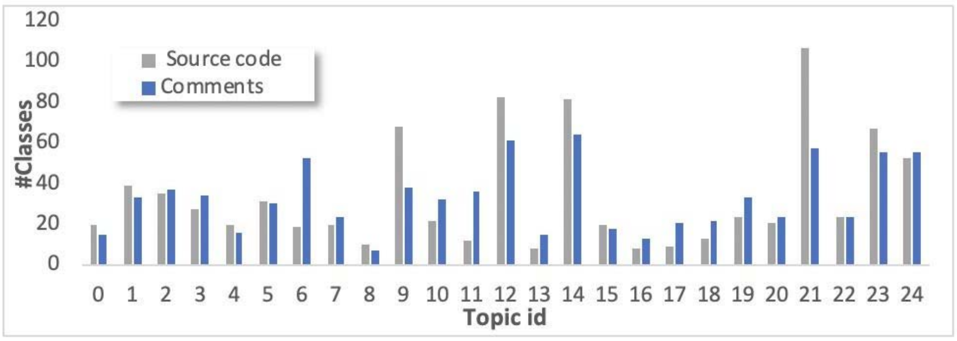
Figure 3, for JFreeChart, highlights the prevalence of topic 21 in the comment's files, indeed, it is retrieved in over 350 classes of the analyzed system, respect to the other topic traced with fewer than 100 classes.

In the case of Spring, the prevalent topic is the 22 that is traced to almost half of the project classes. The other topics, on the other hand, are traced to the source code file with a percentage lower than 10%.

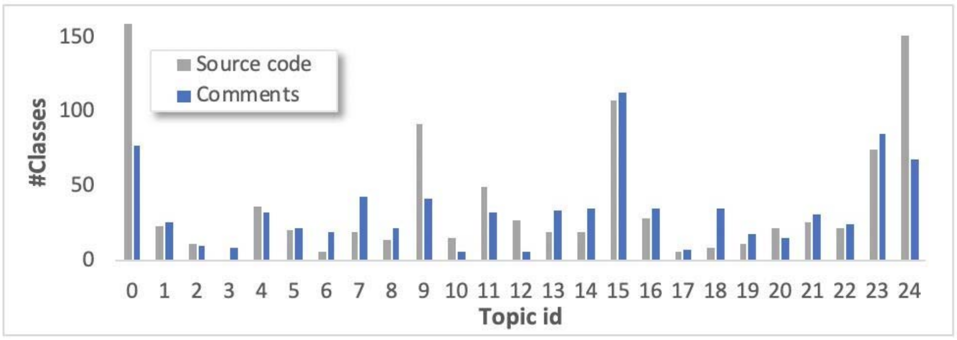
Finally, it is possible to observe that the topics distributions are almost similar for all the software projects analyzed.

Intuitively, this means that there is a high consistency between comments and source code. Furthermore, no topic shows a clear predominance over others.

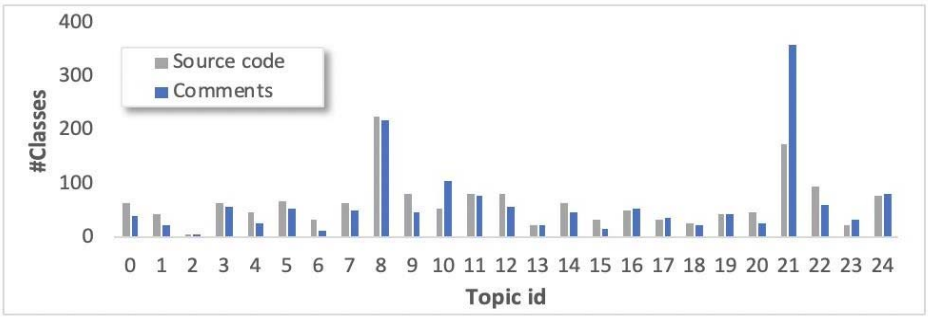
[1] https://pypi.org/project/gensim/



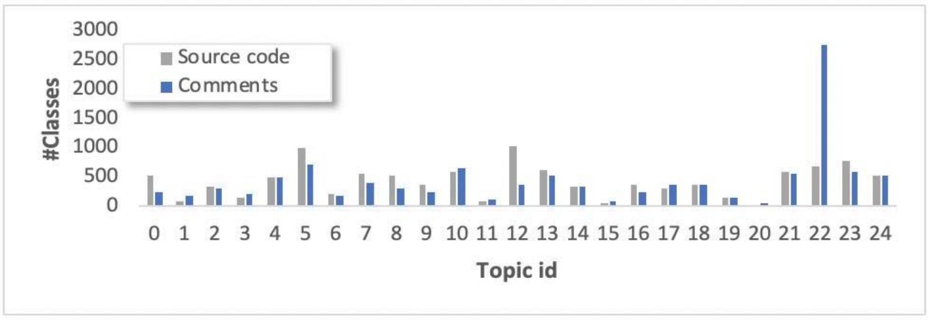
(a) Jedit



(b) Jabref



(c) JFreeChart



(d) Spring

Fig. 3: Comments topics distribution on source code and comments files

RQ2 What is the comments topics incidence per source code files and comment files?

In this analysis, for each class, it is computed the number of comments topics identified in the source code and in the related comment text. To support the analysis a graphical representation is produced using 6 ranges: [0], [1], [2-3], [4-5], [6-7], [8+], to depict, for each system examined, the number of topics generally found within a class.

In particular, for each project, Figure 4 shows a bar graph reporting on the x-axis the topic range, previously established, and on the y-axis the number of classes. As before, also, in this case, the gray bars indicate the number of classes in which the comment topic is found in the comments and the blue bars in the source code. From the results relating to the calculation of the number of classes with a certain number of arguments, it can be observed that in all four projects the files are associated with a maximum of 3 arguments. This means that for all the analyzed projects, a maximum of 3 topics are found in each class, both in the comments and in the source code.

In particular, in the case of Jedit, having 554 classes in the whole system, it is possible to affirm that almost 50% of these contain only one topic within them, and the remainder is related to [2-3] topics, both whether it's comments or source code.

For Jabref, the obtained results are almost similar because also in this case about 45% of the classes are related to one topic or at least [2-3] topics. However, in this case, there is a slightly higher number of classes (about 4 %) with a single topic in the comments, instead almost 20% more classes contain at most [2-3] topics in the source code.

Even in JFreeFChart, the distribution is somewhat similar between the presence of a single topic in the classes and the presence of [2-3] topics. Indeed, 51% of the classes contain at most one topic in the comments, while 48% contain [1-2] topics. Moreover, 44% of the classes contain only one topic; for the source code the difference is slightly greater, 52% [2-3].

Finally, in Spring about 52% of classes contain at most 2-3 topics for both comments and source code, the remaining percentage instead contains only one topic. The case of classes with 0 topics or with more than 4 topics corresponds to a lack of comments in the classes.

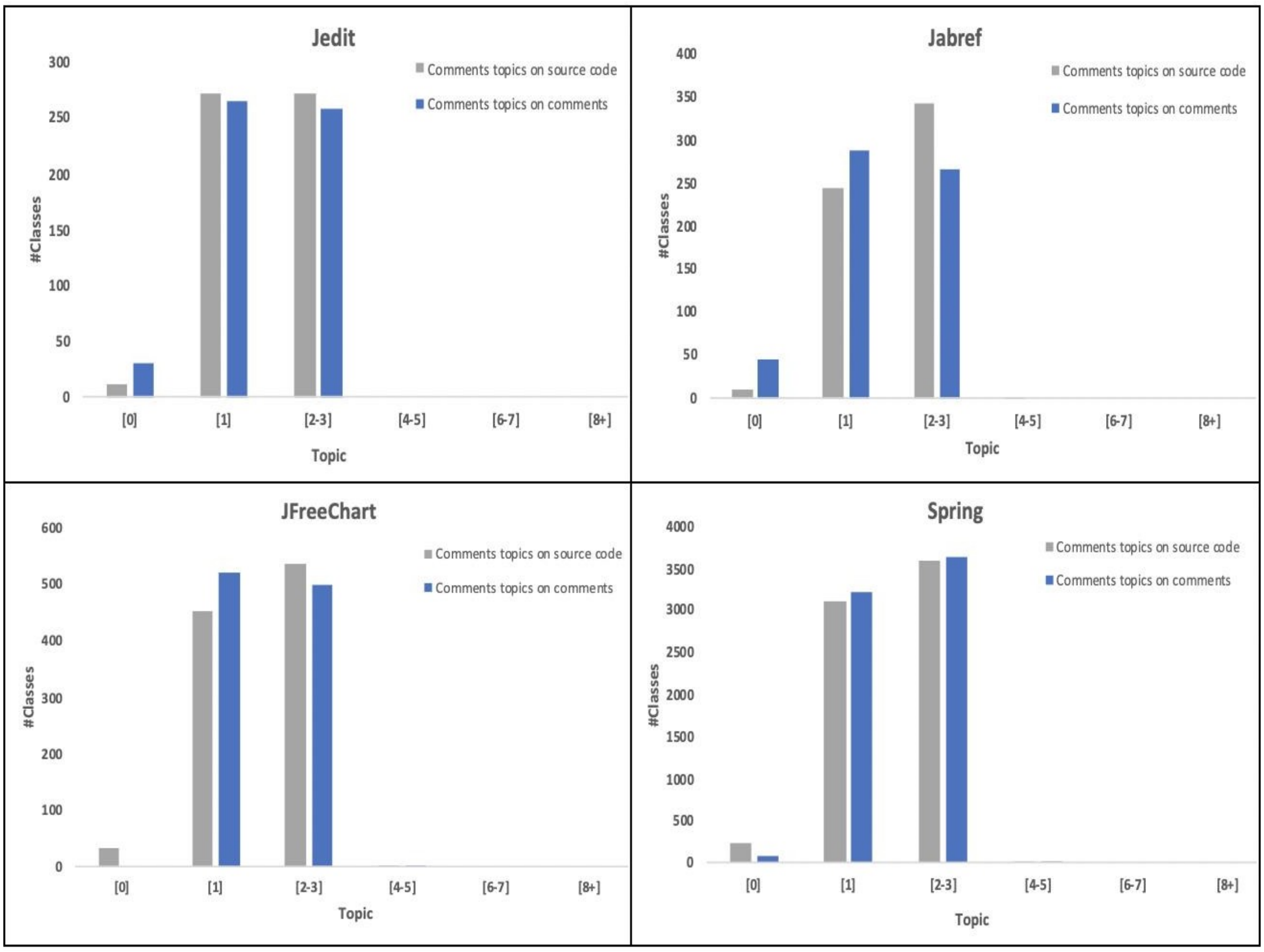


Fig. 4: *Comments topics incidence per source code files and comment files*

VI. Threats to Validity

In this section, the threats to the validity of the proposed research are discussed.

Internal validity: the threat to internal validity concerns whether the results in our study correctly derive from our data. In particular, if the method used for the extraction of the topics is adequate or not. 25 different topics were considered, but these may be insufficient. The use of the LDA model for the extraction of the topics has been preferred to other methods because it is able to overcome the overfitting, and to generating all the connections between the latent topics and the observed words, based on the probability of correlation and the distribution of the words generated by that topic.

External Validity: The threat to external validity is that this study is limited to open source Java projects, therefore one cannot be sure that this method can be applied to projects that are not implemented in Java. Also, there is the generalizability of the results obtained, due to the size of our selected dataset. To mitigate this threat, known systems, in continuous evolution, characterized by different sizes, domains, times, numbers of versions have been considered.

VII. Conclusions

Comment plays an important role in program development and comprehension. However, it is seldom straight-forward to track relations between comments and source code entities algorithmically. The topics model is a technique able to discover semantic elements over a large collection of documents. It received increasing popularity in recent years and has been successfully applied in a variety of domains.

Using the analysis of the source code and code comment it was possible to extract the topics, in order to identify the topic models of the analyzed software projects.

Considering that the comments in the code represent a main source for the system documentation. This simplify the work of software developers from the point of view of code traceability, for maintenance or development, but also for the use of the code itself as a library or framework in other projects. Although many software developers consider comments to be important, existing approaches to software quality analysis completely ignore system comments and focus only on quantitative analysis.

This work presents an approach for the quality analysis and evaluation of comments in the code. Then a precise model is provided to define the quality of the comments in terms of consistency, because the comments should be consistent throughout the whole project, and fully document the system. These should, in fact, be in line with the code and should contribute to the understanding of the system.

From the study of the experimental results, it's possible to conclude that the approach used in this work represents an promising model of analysis of the quality of the comments from the point of view of consistency with respect to the source code.

References

[1] G. Antoniol, G. Canfora, A. de Lucia, and G. Casazza. InformationRetrieval Models for Recovering Traceability Links between Code and Documentation. In Proceedings of the International Conference on Software Maintenance, ICSM '00,page 40. IEEE Computer Society, 2002.

[2] Beat Fluri, Michael Wursch, and Harald C. Gall. Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes. In Proceedings of the 14th Working Conference on Reverse Engineering, WCRE '07,pages 70–79, 2007.

[3] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in Proceedings of the 32nd ACM/IEEE International Conferenc on Software Engineering-Volume 1, 2010, pp. 95–104.

[4] Wei Emma Zhang, Quan Z. Sheng, Ermyas Abebe, Muhammad Ali Babar, Andi Zhou, Mining Source Code Topics Through Topic Model and Words Embedding, 2016. In Proceedings of the 12th International Conference, ADMA 2016, Gold Coast, QLD, Australia, December 12-15, 2016.

[5] Tan,L,Yuan, D.,Krishna,G., Zhou,Y.,2007./ ∗iComment: bugs or bad comments?∗ In: ACM SIGOPS Operating Systems Review, vol.41.ACM,pp.145–158.

[6] McBurney, P. W, McMillan,C. 2016.An empirical study of the textual similarity between source code and source code summaries_ Empir. Softw. Eng.21(1),17–42.

[7] McBurney, P. W, McMillan C. , 2014, Automatic documentation generation via source code summarization of method context. In: proceedings of the 22nd International Conference on program Comprehension, ACM, PP. 279-290.

[8] Huanchao Chen, Zhiyong Liu, Xiangping Chen; Fan Zhou; Xiaonan Luo, Automatically Detecting the Scopes of Source Code Comments, in: 2018 IEEE 42nd Annual Computer Software and Applications Conference, COMPSAC

[9] Woodfield, S.N., Dunsmore, H.E, Shen, V.Y. ,1981. The effect of modularization and comments on program comprehension. In: Proceedings of the 5th International Conference on Software Engineering. IEEE Press, pp.215-223.

[10] de Souza, S.C.B, Anquetil, N. de Oliveira, K. M., 2005. A study of the documentation essential to software maintenance. In: Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting $ Designing for Pervasive Information. ACM, pp.68-75.

[11] C. S. Hartzman and C. F. Austin, "Maintenance productivity: Observations based on an experience in a large system environment, " ser. CASCON '93, 1993.

[12] T. Tenny, "Program Readability: Procedures Versus Comments, " IEEE Trans. Softw. Eng., vol. 14, no. 9, 1988.

[13] S. N. Woodfield, H. E. Dunsmore, and V. Y. Shen, "The effect of modularization and comments on program comprehension, " ser. ICSE '81, 1981.

[14] M. J. B. García and J. C. G. Alvarez, "Maintainability as a Key Factor in Maintenance Productivity: A Case Study, " ser. ICSM '96, 1996.

[15] P. Oman and J. Hagemeister, "Metrics for Assessing a Software System's Maintainability, " ser. ICSM '92, 1992.

[16] M. L. V. D. Vanter, "The documentary structure of source code", Information and Software Technology, vol. 44, no. 13, pp. 767-782, October 2002.

[17] D. Spinellis, Code Quality-The Open Source Perspective, Addison-Wesley, 2006.

[18] T. Tenny, "Program readability: Procedures versus comments", IEEE Trans. Software Eng., vol. 14, no. 9, pp. 1271-1279, 1988.

[19] J. L. Elshoff, M. Marcotty, "Improving computer program readability to aid modification", Communications of the ACM, vol. 25, no. 8, pp. 512-521, 1982.

[20] A. Lakhotia, "Understanding someone else's code: Analysis and experience", Journal of Systems and Software, vol. 23, no. 3, pp. 269-275, 2003.

[21] Daniela Steidl Benjamin Hummel Elmar Juergens , Quality Analysis of Source Code Comments , in: 2013 21st International Conference on Program Comprehension (ICPC),2013.

[22] Blei, D.M., A.Y. Ng, and M.I. Jordan, Latent dirichlet allocation. J. Mach. Learn. Res., 2003. 3: p. 993-1022.