JSEA: A Program Comprehension Tool Adopting LDA-based Topic Modeling

Tianxia Wang
School of Software Engineering
Tongji University
China

Yan Liu
School of Software Engineering
Tongji University
China

***Abstract*—Understanding a large number of source code is a big challenge for software development teams in software maintenance process. Using topic models is a promising way to automatically discover feature and structure from textual software assets, and thus support developers comprehending programs on software maintenance. To explore the application of applying topic modeling to software engineering practice, we proposed JSEA (Java Software Engineers Assistant), an interactive program comprehension tool adopting LDA-based topic modeling, to support developers during performing software maintenance tasks. JSEA utilizes essential information automatically generated from Java source code to establish a project overview and to bring search capability for software engineers. The results of our preliminary experimentation suggest the practicality of JSEA.**

Keywords*—Java program comprehension; Topic models; Interactive tool

I. Introduction

Before performing software maintenance and extension tasks, developers must understand what a program does and how it does [1]. Program comprehension is the activity of understanding how a software system or a part of it works [2]. According to Corbi [3], program comprehension accounts for more than half of the software maintenance time. Researchers and practitioners developed tools that can assist developers in program comprehension, such as JRipples [4], Codecrawler [5], and SonarQube [6]. However, tools applying topic models to program comprehension has not been given as much attention.

With the development of Information Retrieval, topic models appeared as an effective way of extracting semantic information. Topic models are probabilistic models for uncovering the underlying semantic structure of a document collection based on a hierarchical Bayesian analysis of the original texts [7]. Researchers apply topic models to mine source code and get linguistic topics automatically. These topics extract semantic information from programs and tend to correspond to features implemented by the software [8]. Using these topics appropriately, developers should be able to understand programs more effectively.

Researchers have used topic modeling to support varied software engineering tasks, including traceability link recovery [9], concept/feature location [10], source code metrics [11], and many other tasks [12], [13], however, the application of using topic modeling to support interactive program comprehension has not got as much attention. Also, there are gaps between knowing and doing when applying topic modeling to software engineering practice. For instance, feature location is the act of identifying the set of source code fragments in a software system that implement a particular concept, but the boundary lines and definitions of features in programs are vague [14], so it is hard to directly locate features via automatic topic modeling. We need to explore topic modeling and its application on interactive program comprehension further.

In this paper, we explored the source code preprocessing procedure of topic modeling based on the characteristic of source code and introduced JSEA (Java Software Engineers Assistant), an interactive program comprehension tool adopting LDA, which provides a project overview page and a search model. JSEA aims at helping developers learn unfamiliar source code in a faster manner to carry on software maintenance and extension tasks. Considering different programming languages have different characters, in this paper, we focus on Java language, but the concept can be reused for other languages.

This paper makes the following contributions: (1) Based on the characteristic of source code, explore the procedure of source code preprocessing to support topic modeling. (2) Design and develop an interactive program comprehension tool, which extracts semantic information from source code in a useful manner, to support developers during software maintenance. (3) Verify the practicality of JSEA through a preliminary evaluation.

This paper is organized as follows. In the next section, we introduce background. Then, section III detail the preprocess procedure used for JSEA. In section IV and section V, we introduce the interactive program comprehension tool based on LDA and its strategies for the number of topics. In section VI, we provide our evaluation results. Finally, the section **??** concludes and introduces future works.

II. Background

A. Program Comprehension

Some influential theories about program comprehension were proposed in the past, including top-down [15], [16], bottom-up [17], and a combination of the two [18], [19]. Brooks [15] describes top-down theories of program comprehension as hypothesis-driven. When a program is comprehended, the knowledge are organized into distinct domains which bridge between the original problem and the final program. The program comprehension process is reconstructing

knowledge about these domains and the relationship among them. In button-up model, Shneiderman et al. [17] theorizes that programmers first read source code line-by-line and the program comprehension is accomplished by a hierarchical chunking process that organizes several statements into a functional unit. Then, these units can be organized into still higher level units which convey the overall operation of the program. The integrated metamodel of program comprehension have also been proposed [18], [19], in which programmers switch flexibly from top-down to bottom-up comprehension strategies depending on the situation.

Furthermore, Corritore and Wiedenbeck [1] reported that the object-oriented programmers tend to use a strongly top-down approach to program understanding during the early phase of familiarization with the program but use an increasingly bottom-up approach during the subsequent maintenance tasks. Koenemann and Robertson [20] argued that comprehension activities are mostly top-down in a larger program. The tool, JSEA, is aimed at assist developers in comprehending unfamiliar programs, especially large scale programs, so the top-down model for JSEA is suitable. JSEA is also an interactive tool, which facilitates the information exchange between users and the system and then foster program comprehension during software maintenance.

B. Topic Modeling

Topic models were originally developed as a means of automatically indexing, searching, clustering, and structuring large corpora of unstructured and unlabeled documents. Using topic models, topics are extracted from documents and are used to represent the corpora. A topic is a collection of terms that co-occur frequently in the documents of the corpus, so the documents can be clustered by topics and the entire corpus can be indexed and organized in terms of this discovered semantic structure [7], [13]. Latent Dirichlet Allocation (LDA) is a popular probabilistic topic model [21].

C. LDA

Latent Dirichlet Allocation (LDA) is a popular probabilistic topic model. It models each document as a multi-membership mixture of K corpus-wide topics, and each topic as a multi-membership mixture of the terms in the corpus vocabulary. This means that there is a set of topics that describe the entire corpus, each document can contain more than one of these topics, and each term in the entire repository can be contained in more than one of these topics. Therefore, LDA is able to discover a set of ideas or themes that well describe the entire corpus. Blei reported LDA in detail [21]. In this paper, LDA-based topic modeling is used to mine source code and then support developers.

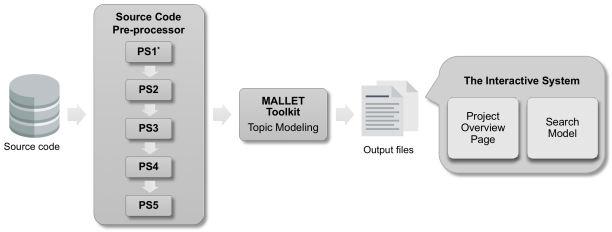
III. Preprocessing Procedure

Before topic modeling, several preprocess steps are generally taken to reduce noise and improve the modeling results [13]. Compared to natural language text, Hindle et al. [22] reported that text extracted from source code is much more repetitive and predictable. Based on the characteristic of source code and prior researches [23], [24], the preprocess procedure was customized through experimentation and brought the following five preprocess steps:

- **Remove programming language information:** Characters related to the syntax of the programming language (e.g., &&, $\rightarrow$) are removed; programming language keywords (e.g., *if*, *while*) are removed. In this paper, a Java language keywords list was customized referring to Oracle official documentation [25]. At first, the customized Java language keywords list was same as the list that Oracle provided, then this list was used to topic modeling and got a result. Secondly, useless Java language keywords were selected from the result and were added into the list. Then, this process was repeated until the result did not have useless Java language keywords. See Table I for a full list. In future, the Java language keywords list can be generated by machine learning instead of manual processing.
- **Split words:** Identifier names are split into multiple parts based on common naming conventions, such as camel case (oneTwo), underscores (one_two), and dot separators (one.two). According to the research of Grant et al. [24], in this paper, we tried to put original identifier names into the source of topic modeling too. For instance, an identifier name called "addMenu", was first split into "add" and "menu", and then "addmenu", "add" and "menu" were put into the source of topic modeling. However, the results were worse. We suppose the reason is the original words like "addmenu" are not in natural language, so adding original words into assets influences the results for topic modeling. Therefore, in this step, identifier names were just split and the original identifier names were abandoned.
- **Remove stop words:** Common English-language stopwords (e.g., *the*, *it*, *on*) are removed. In this paper, the common English-language stopwords list of MALLET LDA toolkit [26] is used. In future, the common English-language stopwords list can be configurable.
- **Remove copyright information:** For open source code, copyright information almost exists in each document. Topic modeling with such information will generate a topic containing copyright information like *author*, *copyright*, *license*. This kind of information is easy to obtain without the help of topic modeling, and it is useless for developers to get a general understanding of programs, so copyright information need to be removed.
- **Remove nondescript information:** In source code, libraries can be imported and packages can be declared, thus topics extracted from source code can contain some common used library names and package names. However, some library names and package names is useless for developers to acquire a general understanding of the program and even might confuse developers. For example, {*set drawing button* ***org*** *init layout panel components pane variables*} is a topic extracted from JHotDraw [27], and "org" is useless in

TABLE I. THE CUSTOMIZED JAVA LANGUAGE KEYWORDS

abstract array arg assert boolean break byte catch case char class code continue default ddouble do don double else enum error exception exist exists extends false file final finally float for id if implementation implemented implements import instanceof int integer interface interfaces invoke invokes java lead long main method methodname methods native new null object objects overrides package packages param parameters precison println private protected public return returned returns short static string strictfp super switch synchronized system this throw throws transient true try version void volatile while



*PS: Preprocess step (see Sect. III)

Fig. 1. JSEA Overview

this topic. We define this kind of library names and package names as nondescript information and remove them in preprocessing.

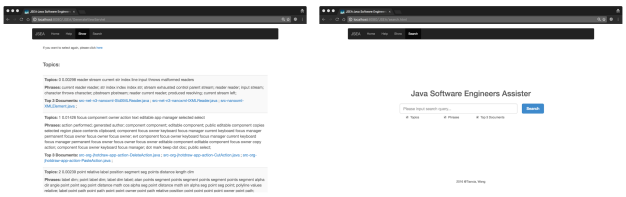
IV. JSEA

JSEA (Java Software Engineers Assistant) is a web application implemented in Java language, setting up in Tomcat server. It infuses topic modeling into program comprehension in an interactive manner, aiming at supporting developers during software maintenance. All the data, results and source code of JSEA is available in Github[1].

A. Design

Fig. 1 provides an overview of JSEA. At the first stage, the Source Code Pre-processor tackles the source code with comments of a Java program, using four preprocess steps mentioned in Sect. III. Then, the data is sent to MALLET LDA Toolkit [26]. MALLET is a Java-based package for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text. Topic modeling based on LDA is completed in this stage. As for the parameters of the model, it affects the practicability of JSEA in a large extent, so the values of parameters were tuned, especially the number of topics (See Section V), through experiments. By default, the parameter settings are as follows: the maximum iteration (–num-iterations) is 1000, the number of most probable words (–num-top-words) is 10, the number of iterations between reestimating dirichlet hyperparameters (–optimize-interval) is 10 and the initial topic model parameters are the default values in the MALLET LDA toolkit [26]. We also allow users to set the value of parameters as their demands. Next, JSEA processes topic modeling results to acquire semantic and structure information about the program as follows:

- **Extracting Topics:** Topic modeling directly generates topics, so JSEA just need to store these topics with index for the following procedure.



(a) The Project Overview Page (b) JSEA-Search

Fig. 2. The Screenshots of JSEA

TABLE II. AN EXAMPLE OF A TOPIC IN THE PROJECT OVERVIEW PAGE

Top Words: 78* 0.00985† *action menu add set bar put item tool actions open*
Phrases: action action; menu item; menu bar; menu add; tool bar; action put action; put action; menu open; action add; menu menu labels
Top 3 Documents:
src-org-jhotdraw-app-DefaultOSXApplication.java-src.txt;
src-org-jhotdraw-app-DefaultMDIApplication.java-src.txt;
src-org-jhotdraw-app-DefaultSDIApplication.java-src.txt

*The index of topics
†The value of Dirichlet parameters

- **Applying Descriptive Phrases to Topics:** When using MALLET LDA toolkit to topic modeling, the LDA toolkit automatically summarizes related phrases for each topic. For example, {*stroke color width fill grow basic font line shape factor*} is a topic extracted from JHotDraw [27], and the LDA toolkit adds phrases like "fill color", "stroke width" and "basic stroke" to this topic. JSEA links these descriptive phrase with their related topics, in order to increase the readability of topics.
- **Applying Documents to Topics:** The probabilistic topic distributions of documents are generated by topic modeling. Using the distributions, JSEA assigns top related documents to each topic.

Finally, JSEA utilizes these information to establish a system allowing developers to explore the programs in an interactive manner, including:

- **A Project Overview Page:** The Project Overview Page is designed for helping developers get an overview of Java projects in a short time. It shows all topics and some related information about the program. Which kind of information will be shown depends on which style is selected by users. JSEA provides four styles: (1) Topics; (2) Topics and Phrases; (3) Topics, Phrases and Top 3 Documents (Recommended); (4) Topics, Phrases and More Top Documents. Note that "More Top Documents" means top 100 related documents for each topic. For example, if users select the third style, the Project Overview Page will be like Fig. 2(a) and Table II gives an example of a topic in Project Overview Page. Users can select a suitable style according to their demands and habits.
- **JSEA-Search:** JSEA-Search (Fig. 2(b)) is a search model, which is designed for satisfying developer's immediate information need. Through JSEA-Search, developers can obtain information semantically and

[1]https://github.com/jseaTool/JSEA

structurally related to the search query. These information can include topics, descriptive phrases, top related documents for each topic and an access to related source code, all of which can help developers quickly determine where to start during performing software maintenance or extension tasks. Users determine searching which kind of information through the checkboxes below the search bar, and all checkboxes are selected as default.

Developers can select suitable sub systems based on their needs. JSEA-Search is more practical than the Project Overview Page, because the latter is helpful just for developers who want to have a general overview of projects in a short time, but MAT-Search can effectively assist developers when they face maintenance or extension tasks.

B. Configuration

JSEA is configured in the following steps: (1) Deploy Tomcat Server; (2) Configure the *web.xml* of JSEA; (3) Configure MALLET LDA toolkit.

V. The Number of Topics

The number of topics greatly affects topic modeling results and thus the results of the study [13]. Too many topics assigns related words to different topics but brings more meaningful information, while too few topics rarely brings related words into different topics but leads to results containing less meaningful information. Therefore, the number of topics need to be estimated.

A few articles proposed approaches determining the number of topics [28], [29], but they were task-specific. Our paper explore the application of topic modeling in a generic perspective other than a task-driven style, so we need to be groping for other estimating approach. Chen et al. [13] reported that many articles choose an optimal value of K (the number of topics) by testing a range of K values and evaluating each in some way. We decided to use the same strategy.

As for how to evaluate topic modeling results, a Naive Criterion was proposed: (1) Label each topics. The categories include *functionality*, *Java library topic*, *design*, *repetitive*, *multiple meaning* and *useless*. The first three are positive, while the last three are negative; (2) Calculate the percentage of each category for each K value; (3) Compare and analysis all results generated by varied K. The basic idea is that the result is better when the percentage of positive label is higher.

We recommend the users of JSEA estimating the number of topics by testing a range of K values and evaluating each using Naive Criterion. For instance, we used JHotDraw [27], a Java GUI framework, as our learning object. Referring to Grant and Cordy [29], where they think 100 to 200 is the best area for the number of topics of JHotDraw, we tested the number of topics ranging from 50 to 250 in 10 increments and evaluated each result using our Naive Criterion. We found that 80 is the most optimum value for the number of topics of JHotDraw.

TABLE III. Task Assignment

Systems	FLT_1 *	FLT_2	RT_1 †	RT_2
JHotDraw	IDE and JSEA	only IDE	IDE and JSEA	only IDE
MALLET	IDE and JSEA	only IDE	IDE and JSEA	only IDE

*FLT: feature location task
†RT: reuse task

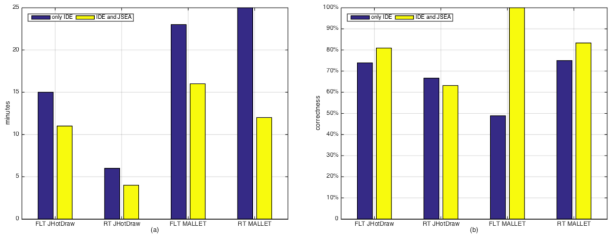


Fig. 3. (a) The time spent on each task; (b) The correctness of each task. FTL means feature location task, and RT means reuse task.

VI. Evaluation

We conducted several experiments on two open-source systems, in order to evaluate the utility of JSEA on software maintenance. The overall metric of our experiments was: Does JSEA save effort for developers when they perform tasks?

One of the open-source systems is JHotDraw version 7.0.6, a Java GUI framework for technical and structured graphics. Another is MALLET version 2.0.8, a Java-based package for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text. JHotDraw version 7.0.6 has 54KLOC (thousands of lines of code), while MALLET version 2.0.8 has 114KLOC, which is 2.11 times larger than JHotDraw version 7.0.6. Note that, 170 is the most optimum value for the number of topics of MALLET, using the same strategy of determining the number of topics with JHotDraw (See Sect. V).

To evaluate the practicality of JSEA, we recruited an experienced Java developer as our volunteer to perform two kinds of software maintenance tasks, and one of authors, who is greatly familiar with both systems and JSEA, was asked to evaluate the correctness of results. As for the two kinds of software maintenance tasks, one is feature location task (FLT), and another is reuse task (RT). For each kind of task, we conducted two tasks for each systems. One is with the help of commonly used IDE (IntelliJ IDEA), another is with the help of IDE and JSEA. Table III summarizes the task assignment. The volunteer performed tasks in Table III from left to right and from JHotDraw to MALLET.

During experimentation, we recorded the time spent on each task and analyzed the correctness of the results. Fig. 3(a) shows the time spent on each task. Fig. 3(b) shows the correctness of the results. The correctness is calculated in the following way: We define the files that considered relevant to each task by the author as *total files*, and define the intersection of *total files* and the files that considered relevant to each task by the volunteer as *correct files*. The correctness is the percentage that dividing the number of *correct files* by the number of *total files*.

In Fig. 3, the blue one means only using IDE, while the

yellow one means using IDE and JSEA. We can see from the left bar chart that the yellow bar is lower than the blue bar for each pair of bars, especially the third pair and the fourth pair. It means JSEA can save time for developers when they perform feature location tasks and reuse tasks. Besides, JSEA is more suitable to support developers on larger scale programs. In the right bar chart, for each pair of bars, the yellow bar is higher than the blue bar or is similar with the blue bar. It means using JSEA do not affect the correctness of the results. In conclusion, JSEA can help developers comprehend programs and then perform tasks faster.

VII. Conclusion and Future Work

In this paper, we explored the source code preprocessing procedure of topic modeling based on the characteristic of source code and developed JSEA (Java Software Engineers Assistant), an interactive tool adopting LDA-based topic modeling, to support developers during software maintenance. JSEA utilizes essential information automatically generated from Java source code to establish a project overview and to bring search capability for developers. The preliminary experimentation indicates that JSEA can effectively help developers comprehend programs, and assist them in maintaining software projects or developing new features with less time.

In future, we plan to integrate topic modeling of MALLET toolkit into JSEA, and calculate the correlation between topics and search queries. Other interesting direction for future work would be to combine the code search system into JSEA, assisting software engineers with more powerful functionality. Besides, we plan to recruit developers with different professional levels as volunteers and repeat the experimentation, in order to verify the practicality of JSEA more effectively.

References

[1] C. L. Corritore and S. Wiedenbeck, "An exploratory study of program comprehension strategies of procedural and object-oriented programmers," *International Journal of Human-Computer Studies*, vol. 54, no. 1, pp. 1–23, 2001.

[2] W. Maalej, R. Tiarks, T. Roehm, and R. Koschke, "On the comprehension of program comprehension," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 4, p. 31, 2014.

[3] T. A. Corbi, "Program understanding: Challenge for the 1990s," *IBM Systems Journal*, vol. 28, no. 2, pp. 294–306, 1989.

[4] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, "Jripples: A tool for program comprehension during incremental change," in *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on*. IEEE, 2005, pp. 149–152.

[5] M. Lanza, S. Ducasse, H. Gall, and M. Pinzger, "Codecrawler-an information visualization tool for program comprehension," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. IEEE, 2005, pp. 672–673.

[6] S. S.A, "SonarQube," https://www.sonarqube.org/, 2017, accessed 27 Feb 2017.

[7] D. M. Blei and J. D. Lafferty, "Topic models," *Text mining: classification, clustering, and applications*, vol. 10, no. 71, p. 34, 2009.

[8] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya, "A theory of aspects as latent topics," in *ACM Sigplan Notices*, vol. 43, no. 10. ACM, 2008, pp. 543–562.

[9] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent dirichlet allocation," *Information and Software Technology*, vol. 52, no. 9, pp. 972–990, 2010.

[10] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining concepts from code with probabilistic topic models," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 2007, pp. 461–464.

[11] Y. Liu, D. Poshyvanyk, R. Ferenc, T. Gyimóthy, and N. Chrisochoides, "Modeling class cohesion as mixtures of latent topics," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009, pp. 233–242.

[12] D. Andrzejewski, A. Mulhern, B. Liblit, and X. Zhu, "Statistical debugging using latent topic models," in *European conference on machine learning*. Springer, 2007, pp. 6–17.

[13] T.-H. Chen, S. W. Thomas, and A. E. Hassan, "A survey on the use of topic models when mining software repositories," *Empirical Software Engineering*, pp. 1–77, 2015.

[14] S. Grant, J. R. Cordy, and D. B. Skillicorn, "Reverse engineering co-maintenance relationships using conceptual analysis of source code," in *2011 18th Working Conference on Reverse Engineering*. IEEE, 2011, pp. 87–91.

[15] R. Brooks, "Towards a theory of the comprehension of computer programs," *International Journal of Man-Machine Studies*, vol. 18, no. 6, pp. 543–554, 1983.

[16] V. Rajlich and N. Wilde, "The role of concepts in program comprehension," in *Program Comprehension, 2002. Proceedings. 10th International Workshop on*. IEEE, 2002, pp. 271–278.

[17] B. Shneiderman and R. Mayer, "Syntactic/semantic interactions in programmer behavior: A model and experimental results," *International Journal of Parallel Programming*, vol. 8, no. 3, pp. 219–238, 1979.

[18] S. Letovsky, "Cognitive processes in program comprehension," *Journal of Systems and Software*, vol. 7, no. 4, pp. 325–339, 1987.

[19] M.-A. Storey, "Theories, methods and tools in program comprehension: Past, present and future," in *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on*. IEEE, 2005, pp. 181–191.

[20] J. Koenemann and S. P. Robertson, "Expert problem solving strategies for program comprehension," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1991, pp. 125–130.

[21] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[22] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 837–847.

[23] S. Thomas, "Mining unstructured software repositories using ir models," 2012.

[24] S. Grant, J. R. Cordy, and D. B. Skillicorn, "Using heuristics to estimate an appropriate number of latent topics in source code analysis," *Science of Computer Programming*, vol. 78, no. 9, pp. 1663–1678, 2013.

[25] Oracle, "Java Language Keywords," http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html, 2015, accessed 2 Aug 2016.

[26] A. K. McCallum, "MALLET: A Machine Learning for Language Toolkit - Topic Modeling," http://mallet.cs.umass.edu/topics.php, 2002, accessed 21 May 2016.

[27] W. Randelshofer, "JHotDraw as Open-Source Project," http://www.jhotdraw.org/, 2007, accessed 27 Feb 2017.

[28] B. Dit, A. Panichella, E. Moritz, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "Configuring topic models for software engineering tasks in tracelab," in *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*. IEEE, 2013, pp. 105–109.

[29] S. Grant and J. R. Cordy, "Estimating the optimal number of latent concepts in source code analysis," in *Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on*. IEEE, 2010, pp. 65–74.