International Journal of Software Engineering and Its Applications
Vol. 9, No. 5 (2015), pp. 43-52
http://dx.doi.org/10.14257/ijseia.2015.9.5.05

Software Evolution via Topic Modeling: An Analytic Study

Shadi Banitaan[1] and Mamdouh Alenezi[2]

[1]*Computer Science and Software Engineering, University of Detroit Mercy, USA*
[2]*College of Computer & Information Sciences, Prince Sultan University, Saudi Arabia*
banitash@udmercy.edu, malenezi@psu.edu.sa

Abstract

Open-source projects continue to evolve resulted in so many versions. Managing, visualizing and understanding an evolving software system are challenging tasks. In this work, we apply Latent Dirichlet Allocation (LDA) to discover source code topics and study their evolution over multiple software versions. We apply LDA to all versions of the system together and then determine how the assignment metric evolves over time. We study the software evolution of two large open-source projects, JEdit and JHotDraw, over twelve versions. The results show that changes in topics across versions are due to actual software changes such as adding, updating, and removing features. Our work suggests that using LDA can open many paths in software evolution research.

Keywords: *software evolution, topic modeling*

1. Introduction

In recent years, many researchers apply information retrieval (IR) techniques to solve software engineering tasks. Two dominant IR techniques have been widely used in the software engineering community namely Latent Semantic Indexing (LSI) [1] and Latent Dirichlet Allocation (LDA) [2]. LSI is based on the principle that terms that are used in the same contexts tend to have similar meanings. It has the ability to extract the conceptual content of textual documents by forming associations between words that occur in similar contexts. LDA is a generative model that estimates the distributions of topics from textual documents. It assumes that documents have been generated using the probability distribution of topics. LDA is able to identify topics in text corpus. Some of the software engineering tasks that have been addressed using IR techniques are bug localization [3], feature location [4], software evolution [5] and traceability link recovery [6].

Most open source software systems continue to evolve resulted in so many versions. Nowadays, we have access to software repositories that were born more than ten years ago. Managing these huge versions is not an easy task. Previous studies showed that the cost of software maintenance and evolution ranges from 50 to 90% of total software cost [7]. Monitoring, visualizing and understanding software evolution are challenging tasks. Recent work used topic modeling to understand how topics evolve over time [8]. Two main methods can be used to discover topic evolution [8]. The first method discovers the topics of each version and then calculates the similarities between them while the second method applies LDA to all versions of the system together and then determines how topic metrics evolve over time.

Although LDA was effectively used in both natural language processing community and software engineering community, it did not always yield the expected results on software artifacts [6]. This is because software engineering researchers used the same parameter values as the values used for natural language text. On the same hand, LDA requires choosing the number of topics (K). It is still not yet clear how to set the number

ISSN: 1738-9984 IJSEIA

of topics in order to apply LDA for software evolution. In this paper, we explore the task of using LDA to support software evolution. We examine the project history of two open source projects namely JEdit and JHotDraw. To set the number of topics, we used the approach proposed by Arun et al. [9].

The rest of the paper is organized as follows. Section 2 presents some background information about LDA and Arun et al. measure [9]. Section 3 presents the related work. Section 4 describes the approach and presents two case studies. Section 5 discusses the threats to validity that could have affected our study. Finally, Section 6 concludes the paper.

2. Background

Latent Dirichlet allocation (LDA) is a probabilistic model used to extract the hidden topics existing in a collection of documents and to represent each document as a finite mixture on the topics [2]. Each topic is a probability distribution over the set of terms of the document collection. In LDA, each document can have several topics, and each topic can exist in more than one document. LDA is able to discover a representation of ideas or themes that describe the corpus as a whole [5]. LDA is a generative statistical model that helps alleviating model over-fitting, compared to other topic models like Probabilistic LSI [10]. In this study, we used the 'topicmodel' package version 0.2-1 in the R language version 3.1.1. The parameters of LDA are chosen based on the recommendation of the literature [10].

In order to estimate topic and word distributions in LDA, the number of topics should be selected first. Arun *et al.*, [9] proposed a measure to find the right number of LDA's topics by looking at distributions generated from topic-word and document-topic matrix outputs of LDA. Formally, for a corpus C, their proposed divergence measure is the following:

$$\text{Measure}(M_1, M_2) = KL(C_{M1} || C_{M2}) + KL(C_{M1} || C_{M2})$$

Where M1 is the topic-word matrix and M2 is the document-topic matrix, C_{M1} is the distribution of singular values of M1, C_{M2} is the distribution obtained by normalizing the vector L*M2 where L is a 1*D vector of lengths of each document. After computing the measure, a plot is generated where the y-axis represents the divergence values when varying the number of topics (the x-axis). Then, the right number of topics is selected where the divergence dips to zero. In this paper, we adopt this measure for selecting the optimal value of the number of topics.

3. Related Work

Identifying topics from software repositories has been the focus of several software engineering researchers. Most of work on that area is focused on comprehending source code, calculating coupling/cohesion metrics [12], categorizing bug reports [13, 14], feature location [15, 16], and finding traceability links between requirements and code [17, 18].

Savage *et al.*, [16] developed a topic visualization tool named ‘TopicXP’, which extracts identifier names and comments from source code using LDA. However, they failed to mention any approach or technique to specify the number of topics (K). Gethers and Poshyvanyk [12] proposed a new coupling metric based on LDA namely the RTC metric. However, they choose K to be 75 without any justification why did they choose this number. Oliveto et al. [6] compared the ability of LDA and different techniques to recover tracability links between code and documentation. However, they varied the number of topics used to study its impact on the recovery accuracy.

Thomas *et al.*, [5] evaluated topic modeling in the analysis of software evolution using two well-documented systems. They computed various metrics on the discovered topics and found that large majority of topic evolutions correspond with actual code change activities by developers such as corrective maintenance, improvements, and the addition of new features. They have applied LDA to explore software evolution, but it is not clear how to set the number of topics. In their case, they just fixated the number of topics to 45 based on a previous study. In this work, a very simple and effective approach is used to set the number of topics which will eventually help researchers to easily specify K in software engineering tasks.

Two different studies have explored the possibility of determining the tuning parameters of LDA in mining software artifacts context. Grant and Cordy [19] proposed a heuristic-based technique to tackle the challenge of determining the optimal number of LDA topics for a source code corpus of methods by taking into consideration the location of these methods in files or folders, along with the conceptual similarity between methods. The impact of topic count on the models relative quality is unmistakable from their study. However, their approach is a heuristic-based method which cannot be generalized to different contexts. Panichella *et al.*, [20] employed a genetic algorithm approach to identify suitable hyper-parameter values of LDA models. They map the LDA output to graphs and then cluster these graphs. By calculating several clustering metrics, the quality of these clusters is then used to judge the quality of the LDA model.

4. Approach and Case Study

In this section, we investigate the use of LDA to support software evolution. We describe the approach that is used to identify the topics from source code. After that, we present two case studies. We also analyze and visualize the results.

4.1. Terminology

The vocabulary for explaining LDA is as follows. A word is an item from a vocabulary, a document is a sequence of N words denoted by $d = (w_{1...}w_N)$, and a corpus is a collection of M documents denoted by $D = \{d_{1 ...} d_M\}$. LDA assumes a generative process for generating a document d in a corpus D [2] as follows:

1. Choose $N \sim Poisson(\xi)$.
2. Choose θ $Dir(\alpha)$.
3. For each w_i:
   - Choose a topic $z_n \sim Multinomial(\theta)$.
   - Choose a word w_i from $p(w_i|z_n,\beta)$, a multinomial probability conditioned on z_n.

4.2. Systems Under Study

Two well-documented Java systems have been used in this study. The first system is JEdit. JEdit is an open source text editor written in Java. It is maintained by a world-wide development team. JEdit contains many features such as auto indent, word wrap, and syntax highlighting for more than two hundred languages. We consider twelve versions of JEdit from version 3.0.0 to version 4.2. The second system is JHotDraw. JHotDraw is a medium-sized, 2-D drawing framework that supports the development of customized drawings editors. It is a GUI framework for technical and structured Graphics. It was initially developed as an exercise of good program design and has become a standard system for experiments and analysis in topic and concern mining. We consider twelve versions of JHotDraw from version 5.2.0 to version 7.4.1. Table I shows a summary of the datasets.

Table I. Summary of the Datasets

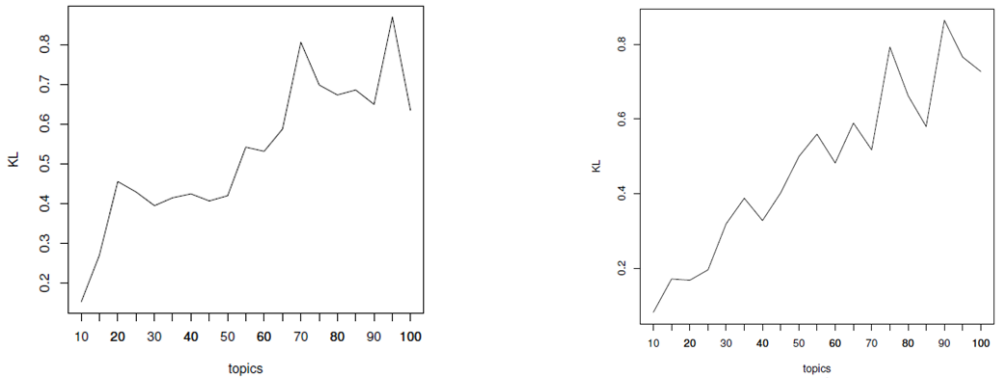
System	JEdit	JHotDraw
Purpose	Text Editor	Drawing Framework
Language	Java	Java
# of Versions	12	12
# of Classes	From 252 to 394	From 309 to 485
Time Period	Dec 2000 to Dec 2004	Feb 2001 to Aug 2010
Source	www.jedit.org	www.jhotdraw.org

4.3. Preprocessing

In this work, we applied the approach of Linstead *et al.*, [21] where they applied LDA to all documents of all the versions at once. One advantage of this approach is that no constraints are placed on the evolution of topics, which results in a great flexibility for describing large corpus, which is usually the case of open source systems.

For both case studies, comments, identifiers, and string-literals were extracted from the software systems and used to create the document collections for each system. Several preprocessing steps are then applied to these documents, these steps are required by any information retrieval technique [22]. There steps are explained as follows:

- Filtering: syntax and programming language keywords are filtered out.
- Tokenization: each word is then tokenized based on common naming practices, such as camel case (firstName) and underscores (first_name)
- Stop-words Removal: common English terms are removed (stop words) to eliminate noise.
- Stemming: the process of conflating the variant forms of a word into a common representation (*e.g.*, "changing" becomes "chang").
- Pruning: the number of terms that can end up the bag-of-words is very large which usually would cause a problem in most text-mining applications. To select the most useful subset, a filter has been applied to remove the overly common terms that appear in too many documents (=80%), as they can be seen as a non-informative and background terms.



(a) JEdit (b) JHotDraw

Figure 1. Number of Topics versus KL Divergence

4.4. Selecting the number of topics

Based on the adopted measure for selecting the number of topics, the numbers of topics were set to 30 and 20 for JEdit and JHotDraw respectively. These values have been selected because the KL divergence dips down close to zero when the number of topics is 30 for JEdit and 20 for JHotDraw (Figure 1).

4.5. The Generated Topics

We calculated the assignment metric and analyzed the discovered topics. Formally, the assignment of a topic z_n at version V_i is the summation of the membership values of all documents in Version V_i [8].

$$A(z_{n,}V_i) = \sum_{j=1}^{|V_i|} d_{ji}(\theta_k)$$

Since the numbers of topics are 30 and 20, we show the analysis of few selected topics. Table 2 shows three selected topics, their labels, their top five terms, and their top corresponding document, and their top two corresponding versions based on the assignment values for each system. The values inside the parenthesis indicate the membership values of documents and the assignment values of versions. The table indicates that the top terms in each topic are coherent and much related to one concept and they are semantically similar. Moreover, each topic corresponds and fit with the top-matching document. For JEdit, the assignment values for these topics are slightly larger in the last two selected versions (version 4.1 and version 4.2).

Table 2. Selected Topics in JEdit and JHotDraw

System	Topic	Label	Top 5 Terms	Top Document	Top 2Versions
JEdit	6	Editing	color, font, highlight, line, style	Gutter.java (0.75)	4.2(10.05), 4.1(8.89)
	10	Browsing	jpanel, actionhandl, emptybord, cancel, jbutton	ToolBarOptionPane.java (0.62)	4.2(17.79), 4.1(17.23)
	15	Searching	search, view, buffer, replac, start	SearchAndReplace.java (0.82)	4.2(8.76), 4.1(7.62)
JHotDraw	12	XML	valu, object, param, name, key	SVGInputFormat.java (0.88)	7.0.9(22.45), 7.1.0(22.99)
	14	Drawing	color, editor, label, drawingeditor, attribute	ButtonFactory.java (0.917)	7.0.9(28.41), 7.0.8(28.54)
	17	Display	view, action, file, project, applic	DefaultOSXApplication.java (0.762)	5.4.b2(36.24), 7.1.0(40.00)

To better understand the results, we use visualization. Figures 2 and 3 depict the heatmap of the assignment evolutions of JEdit and JHotDraw. The colors represent the assignment values for topics in each version of the system, where darker colors (blackish) indicate higher assignment values. The following subsections analyze the heatmap view of JEdit and JHotDraw respectively.

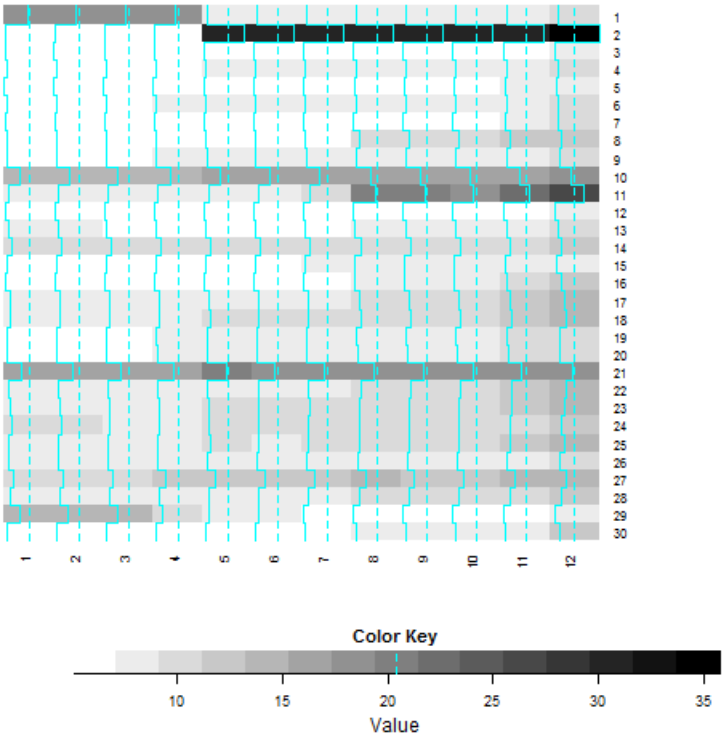


Figure 2. Heatmap View of JEdit Topics

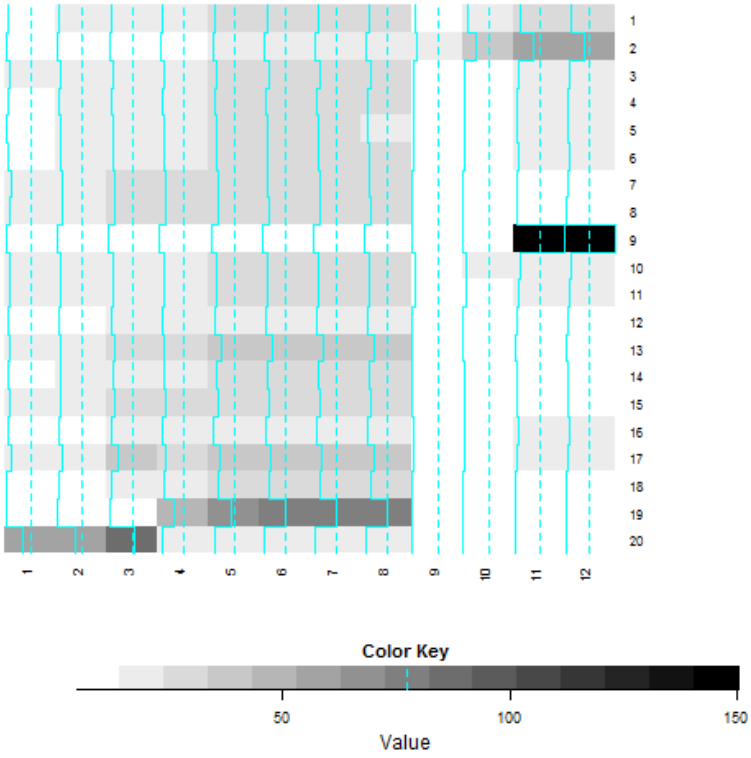


Figure 3. Heatmap View of JHotDraw Topics

4.6. JEdit Heatmap

We notice that some topics born in later versions of the system (*e.g.*, topic 2 was born in version 5) while other topics appear in early versions and then disappear (*e.g.*, topic 29 was born in version 1 and almost died after version 6). We notice that many topics have darker colors in version 12(4.2) which indicate that some significant additions have been made at this version (*e.g.*, new functionalities, refactoring, or a new release). Figure 4 shows the features that have been updated in version 4.2.

```
- Added Factor syntax highlighting.

- Updated BBj syntax highlighting (Thomas Bock).

- Added ANTLR syntax highlighting (Brant Gurganus).

- Added D syntax highlighting (Jim Lawton).

- Added INNO setup syntax highlighting (Ollie Rutherfurd).

- Added Doxygen syntax highlighting. This includes doxyfile*
  configuration files, and Doxygen comments in C, C++, Objective-C and
  D modes. (Jim Lawton)
```

Figure 4. Updated Features in version 4.2 of JEdit

4.7. JHotDraw Heatmap

We notice that some topics born in later versions of the system (*e.g.*, topic 9 was born in version 11) while other topics appear in early versions and then disappear (*e.g.*, topic 8 was born in version 1 and almost died after version 8). We notice that many topics have darker colors in version 7.0.7 which indicates that some significant additions have been made at this version (*e.g.*, new functionalities, refactoring, or a new release). Figure 5 shows the features that have been updated in version 7.0.7.

- Reorganized file structure.
- Fixed NetBeans project files and Ant build scripts.
- Fixed guide.

Figure 5. Changes in Version 7.0.7 of JHotDraw

After version 8 (7.1.0), a lot of changes have been introduced which explains the radical changes in the topics distribution after that version. Figure 6 shows the changes in version 9 (7.2.0).

Drawing Framework:

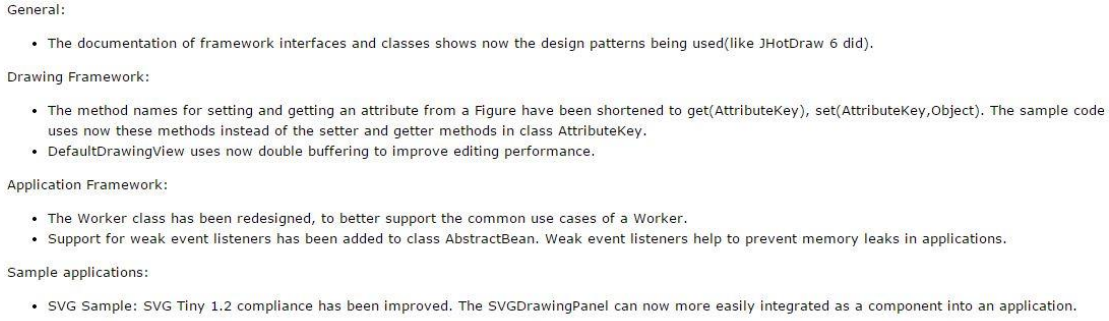
- The type safety of figure attributes has been improved, by making use of type tokens in AttributeKey and setter and getter methods with templates in Figure
- The usability has been improved. Figures indicate there bounds when the mouse is moved over them. A substantial number of user interface classes for the creation of toolbars have been added, and are used by the SVG sample application.
- Drawing objects can now be serialized using the SerializationInputOutputFormat. This allows to easily implement short-term persistence for drawings (for example for clipboard support).

Application Framework:

- The resource support has been reimplemented to be closer to JSR-296. Resource bundles can now use place holders in their values. The resource keys used by JHotDraw match now with the correspoinding JavaBeans property names.

Figure 6. Changes in Version 7.2.0 of JHotDraw

Topic 20 has high values in versions 1 to 3. Topic 19 has higher values from version 4 to version 8. The reason behind that is that version 4 gone through a major package restructuring. The new package structure of "org.jhotdraw" has been introduced instead of the old "CH.ifa.draw" package structure. Also, in later version, the copyright notices have been modified and updated in all source code file (Topic 20). Topic 9 has very high values in the last two versions. In version 11 (7.3.0), many changes have been introduced (Figure 7).


General:

- The documentation of framework interfaces and classes shows now the design patterns being used(like JHotDraw 6 did).

Drawing Framework:

- The method names for setting and getting an attribute from a Figure have been shortened to get(AttributeKey), set(AttributeKey,Object). The sample code uses now these methods instead of the setter and getter methods in class AttributeKey.
- DefaultDrawingView uses now double buffering to improve editing performance.

Application Framework:

- The Worker class has been redesigned, to better support the common use cases of a Worker.
- Support for weak event listeners has been added to class AbstractBean. Weak event listeners help to prevent memory leaks in applications.

Sample applications:

- SVG Sample: SVG Tiny 1.2 compliance has been improved. The SVGDrawingPanel can now more easily integrated as a component into an application.

Figure 7. Changes in Version 7.3.0 of JHotDraw

The aforementioned changes have led to a lot of changes in both Javadoc documentation and test cases. If we examine the top terms in Topic 9, we will see that we have junit, javadocmethod, test, exception, framework, and testcas.

Both topics 7 and 8 die after version 8. In the changes of version 9, drawing objects can now be serialized using the SerializationInputOutputFormat. This allows to easily implement short-term persistence for drawings (for example for clipboard support). A substantial number of user interface classes for the creation of toolbars have been added, and are used by the SVG sample application. For these reasons, both topics 7 and 8 died after version 8. These words in these topics are no longer exist in the system vocabulary because of these changes.

Topic 2 has high values in the last three versions. The components structure and the gui components were majorly changed in these last versions. For instance, DefaultDrawingView uses in the last three versions double buffering to improve editing performance. Some classes were redesigned to better support the common use cases.

5. Threats to Validity

This study has been performed using only two case studies. The two case studies are quite similar. They are both long-lived projects and large open-source software systems. The results may vary if there is no enough history in order to generate meaningful LDA models (i.e., if the projects are not long lived). The size of the system is another important factor where the issues that arise in a large project are not the same in a small project. We think that this methodology could be also applied to other large, long-lived, open-source software projects, but we have not tested the results with other case studies. These two systems were chosen because they are well-documented and well-designed which allows us to validate and test the proposed approach.

6. Conclusion

In this paper, we investigated the use of LDA to support software evolution. LDA has been applied to all documents of all versions at once. We used the approach proposed by Arun to set the number of topics. The assignment metric has been calculated and visualization has been used to understand topic evolution. We examined the project history of two large open-source projects, JEdit and JHotDraw. The experiments showed

that changes in topics across versions are due to actual software changes such as addition of new features, updating of features, and removing features. As a result, using LDA facilitates the software evolution task. Future directions include using and defining other metrics on the output of LDA. In addition, we would like to perform more case studies on other systems.

References

[1] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," (1990) JASIS, vol. 41, no. 6, pp. 391–407.
[2] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," (2003) the Journal of machine Learning research, vol. 3, pp. 993–1022.
[3] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," (2012) in 34th International Conference on Software Engineering (ICSE) IEEE, 2012, pp. 14–24.
[4] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," (2013) Journal of Software: Evolution and Process, vol. 25, no. 1, pp. 53–95.
[5] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Studying software evolution using topic models," (2014) Science of Computer Programming, vol. 80, pp. 457–479.
[6] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia. "On the equivalence of information retrieval methods for automated traceability link recovery,". (2010) in IEEE 18th International Conference on Program Comprehension, pp. 68–71.
[7] R. C. Seacord, D. Plakosh, and G. A. Lewis, "Modernizing legacy systems: Software technologies," (2003) Engineering Process and Business Practices.
[8] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Validating the use of topic models for software evolution," (2010) in 10th IEEE Working Conference on Source Code Analysis and Manipulation, pp. 55–64.
[9] R. Arun, V. Suresh, C. V. Madhavan, and M. N. Murthy, "On finding the natural number of topics with latent dirichlet allocation: Some observations," (2010) in Advances in Knowledge Discovery and Data Mining.Springer, pp. 391–402.
[10] T. Hofmann, "Probabilistic latent semantic indexing,", (1999) in Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. ACM, pp. 50–57.
[11] T. L. Griffiths and M. Steyvers, "Finding scientific topics,", (2004) Proceedings of the National academy of Sciences of the United States of America, vol. 101, no. Suppl 1, pp. 5228–5235.
[12] M. Gethers and D. Poshyvanyk, "Using relational topic models to capture coupling among classes in object-oriented software systems,", (2010) in IEEE International Conference on Software Maintenance, pp. 1–10.
[13] K. Somasundaram and G. C. Murphy, "Automatic categorization of bug reports using latent dirichlet allocation,", (2012) in Proceedings of the 5th India Software Engineering Conference, pp. 125–130.
[14] N. Pingclasai, H. Hata, and K.-i. Matsumoto, "Classifying bug reports to bugs and other requests using topic modeling,", (2013) in Software Engineering Conference, pp. 13–18.
[15] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining eclipse developer contributions via author-topic models,", (2007) in Fourth International Workshop on Mining Software Repositories.
[16] T. Savage, B. Dit, M. Gethers, and D. Poshyvanyk, "Topic xp: Exploring topics in source code using latent dirichlet allocation,", (2010) in IEEE International Conference on Software Maintenance, pp. 1–6.
[17] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery,", (2011) in 27th IEEE International Conference on Software Maintenance, pp. 133–142.
[18] R. Oliveto, M. Gethers, G. Bavota, D. Poshyvanyk, and A. De Lucia, "Identifying method friendships to remove the feature envy bad smell (nier track),", (2011) in Proceedings of the 33rd International Conference on Software Engineering, pp. 820–823.
[19] S. Grant and J. R. Cordy, "Estimating the optimal number of latent concepts in source code analysis,", (2010) in 10th IEEE Working Conference on Source Code Analysis and Manipulation, pp. 65–74.
[20] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," (2013) in Proceedings of the 2013 International Conference on Software Engineering, pp. 522–531.
[21] E. Linstead, C. Lopes, and P. Baldi, "An application of latent dirichlet allocation to analyzing software evolution," , (2008) in Seventh International Conference on Machine Learning and Applications, 2008, pp. 813–818.
[22] C. D. Manning, P. Raghavan, and H. Sch¨utze, Introduction to information retrieval, (2008) Cambridge university press Cambridge, vol. 1.
[23] S. Haiduc and A. Marcus, "On the use of domain terms in source code,", (2008) in The 16th IEEE International Conference on Program Comprehension, pp. 113–122.

