International Journal of Software Engineering and Its Applications
Vol. 9, No. 1 (2015), pp. 183-190
http://dx.doi.org/10.14257/ijseia.2015.9.1.16

Extracting High-Level Concepts from Open-Source Systems

Mamdouh Alenezi

College of Computer and Information Science, Prince Sultan University, Riyadh 11586, Saudi Arabia
malenezi@psu.edu.sa

Abstract

Analyzing the unstructured information in the source code (that is, the comments and identifiers) is based on the idea that the unstructured information reveals, to some extent, the concepts of the problem domain of the software. This information adds a new layer of source code semantic information and captures the domain semantics of the software. Developers use identifiers, method names, and comments to incorporate components of the solution domain of the software. Topic models reveal topics from the corpus, which embody real world concepts by analyzing words that frequently co-occur. These topics have been found to be effective mechanisms for describing the major themes spanning a corpus. Recently, software engineering researchers established that topic models can be effective in structuring various software artifacts, such as bug reports and requirements documents. In this paper, we extract topic models from the textual content of source code by conducting a case study on the source code of Java-based open-source systems, ArgoUML, Checkstyle, JHotDraw and jEdit. The paper investigates the effectiveness of LDA in comprehending large open-source software systems.

Keywords: *Open source, Source code, LDA, Topic Extraction*

1. Introduction

Program comprehension is one of the essential activities during software maintenance and evolution [1]. Usually, developers spend around 60% of their time working comprehending the system while doing software maintenance tasks [1], especially the source code. Understanding and comprehending a small system is usually not a problem and does not take much time. However, comprehending a large system is essentially not an easy task. Small software can be analyzed by manually examining the code to discover its functional architecture. For large-scale system, software engineers rely on source code analysis techniques, such as, control and data flow and call graphs [2]. Although these techniques are useful and very helpful to understand the system interactions, they don not convey any information regarding the functional intent of the system. Structural relationships between software entities focus on a very low level of granularity of dependencies, such as, function calls. This information does not reveal any underlying functional behavior or purpose of the studied system.

An essential step towards comprehending and understanding the functional behavior of any system is to find and recognize business topics that exist in the source code of the system. These business domain objects are modeled as high level components and then realized in the implementation and transformed into code. For example consider an UML modeling application that models UML diagrams and deals with objects, figures, relation-ships, and cardinality. When a maintainer with no application knowledge wants to add a new feature or modify one of the features, will find it very di cult before comprehending and understanding the main functionality of the application. Extracting business topics from the source code and establishing the relationship between them would be a huge sup-port in finding related data structures, methods, classes. This will eventually help the

ISSN: 1738-9984 IJSEIA

developer or the maintainer productivity, especially when dealing with a large system with little documentation.

One technique to find and discover the source code business topics is to extract the semantic information by analyzing the unstructured information in the source code (that is, the comments and identifiers) is based on the idea that the unstructured information reveals, to some extent, the concepts of the problem domain of the software. Developers usually leave trails of the functional behavior of the system in the comments, identifier names, method names, data types and so on [3, 4]. This paper introduces a Latent Dirichlet Allocation (LDA) based approach for finding source code topics. LDA is a very widespread technique in text document classification and extracting topics from textual documents.

The rest of the paper is organized as follows: Section 2 presents some background information about LDA. Section 3 describes the study methodology. Section 4 presents the experimental evaluation and discussions. Section 5 discusses threats to the validity of the study. Section 6 discusses related work. Section 7 concludes the paper.

2. Latent Dirichlet Allocation (LDA)

LDA is a popular fully generative and probabilistic topic model utilized to excerpt the hidden topics existing in documents collection and to represent each document as a finite mixture over the set of topics [5]. Every topic is a probability distribution over the set of words that compose the vocabulary of the document collection. In the LDA model, each document is a multi-membership combination of topics, which means that each topic can be contained in more than one document and each document can contain multiple topics. LDA is capable of discovering a representation of concepts or ideas that describe the corpus as a total [6].

Latent Dirichlet Allocation (LDA) [5] is a method to construct topic models, an un-supervised machine-learning algorithm to discover topics in a corpus and allocate topics distributions over each document, as well as distributions of words over topics. The reason behind adopting LDA in our study is the fact that LDA is a statistical model that supports alleviating model over-fitting, paralleled to other topic models like Probabilistic LSI [7]. In addition, LDA has been shown to be effective for different software engineering purposes. Examples include using LDA to study the evolution of source code [6], refactor code [8], compute source code metrics [3], categorize bug reports [9], localize features and concerns [10], program comprehension [11], and recover traceability links between source code and requirements documents [12, 13].

In this study, we used the 'topicmodel' package version 0.2-1 in the R language version 3.1.12. The LDA parameters were chosen based on the recommendation of the literature [14]. The used parameters are = 50/K and = 0.01 where K is the number of topics.

3. Study Methodology

3.1. Unstructured Source Code

When analyzing source code, traditional techniques rely on the structured data available in source code (*e.g.*, syntax, program semantics, and control-flow). Aside from the structured data, the source code contains a large amount of unstructured data. The comments in the source code are used to indicate the domain information of the software in the form of developer messages and descriptions. Identifier names include package names, class names, method names, and local and global variable names which represent parts of the domain problem. String literals also can be found in print commands and functions. Without the utilization of the source code structured information, these

unstructured data can help comprehend the high-level functionality or concept of the source code [15].

3.2. Systems Under Study

We perform thorough case studies on Java-based well-known open-source software sys-tems, ArgoUML, Cechstyle, JHotDraw and jEdit. Table 1 shows Characteristics of the systems under study. ArgoUML is a large-sized, a UML modeling tool that includes sup-port for standard UML diagrams. It was initially developed by a PhD student at University of California, Irvine. Checkstyle is a medium-sized, development tool that checks whether code adheres to a coding standard. It automates the process of checking Java code resulting in coding standard enforcement. JHotDraw is a medium-sized, 2-D drawing framework that supports the development of customized drawings editors. It is used to model technical and structured Graphics with a GUI framework. It was initially developed as an application of good program design. jEdit is a medium-sized, text editor. It focuses on providing different features for developers, including macro scripting, syntax highlighting, and a comprehensive plug-in environment.

Table 1. Characteristics of the Systems Under Study

Project	Version	# Files
ArgoUML	0.34	1929
Checkstyle	5.7	820
JHotDraw	7.4.1	585
jEdit	5.1.0	572

3.3. Study Setup

String-literals, identifiers, and comments were excerpted from the software systems and utilized to create the documents for each system. An example is shown in Figure 1 to clarify the preprocessing of the unstructured source code data. A number of preprocessing steps are then applied to the source code [6]. Figure 2 shows a high-level visualization of the preprocessing steps. These preprocessing steps are common in most information retrieval techniques [16]. First, syntax and programming language keywords are filtered out. Second, each word is then tokenized according to well-known naming practices, for instance, underscores (first_name) and camel case (firstName). Third, common English terms are removed (stop words) to eliminate noise. The final step is to prune the vocabulary. 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. In order to select the most useful subset, a filter has been applied to remove the overly common terms that appear in too many documents (=90%), as they can be seen as a non-informative and background terms. Table 2 shows the number of terms for each system after preprocessing.

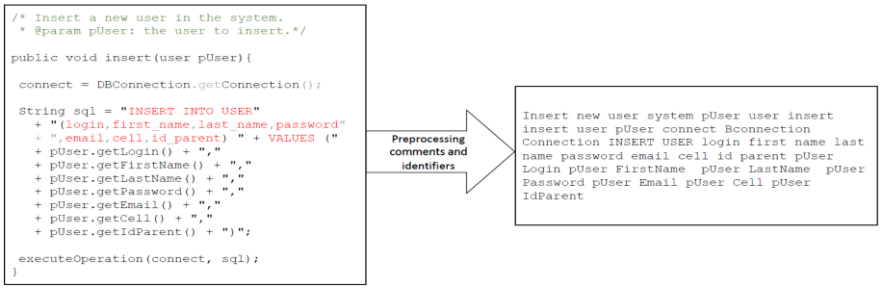

Figure 1. An Example of Unstructured Source Code Preprocessing

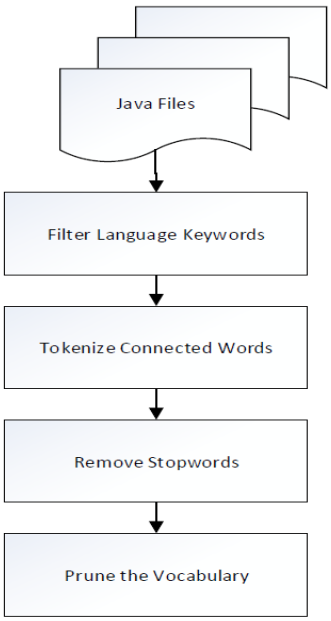

Figure 2. Preprocessing Steps

Table 2. Number of Terms

Project	# Terms
ArgoUML	58994
Checkstyle	17818
JHotDraw	24529
jEdit	41001

3.4. Choosing K

Choosing how many topics to use in LDA is still a research problem. Finding the best number of topics is still a problem and it is not only for the source code domain. Topic excerption in textual documents encounters the same problem. In this work, we adopted a well-known approach for determining the number of topics [17]. Their approach specifies that the number of topics T can be determined by running LDA for different values of T with freezing the LDA hyper-parameters. For each value of T, they estimate the symmetric Kullback-Leibler divergence [18] between the singular values of the topic-word matrix and the document-topic matrix using the following equation:

$$\text{Measure} = KL(CM1||CM2) + KL(CM2||CM1) \quad (1)$$

Their approach calculates the symmetric Kullback-Leiber divergence of the Singular value distributions of of two matrices M1 and M2. In this equation, CM1 represents the singular values distribution of the topic word matrix, CM2 represents the distribution obtained by normalizing the vector L*M2 where L is a one-dimensional vector of documents lengths in the corpus and M2 is the document topic-matrix. To determine T, choose T where the minimum value of the measure is. It is noteworthy that these distributions CM1 and CM2 are in sorted order.

Figure 3 plots the KL divergence measure against a varying number of topics (5-35) for the studied systems. The curve dips down where T is 6 in ArgoUML, dips down where T is 5 in Checkstyle, dips down where T is 5 in JHotDraw, and dips down where T is 10 in jEdit. We selected T based on the minimum value of the KL divergence measure [17].

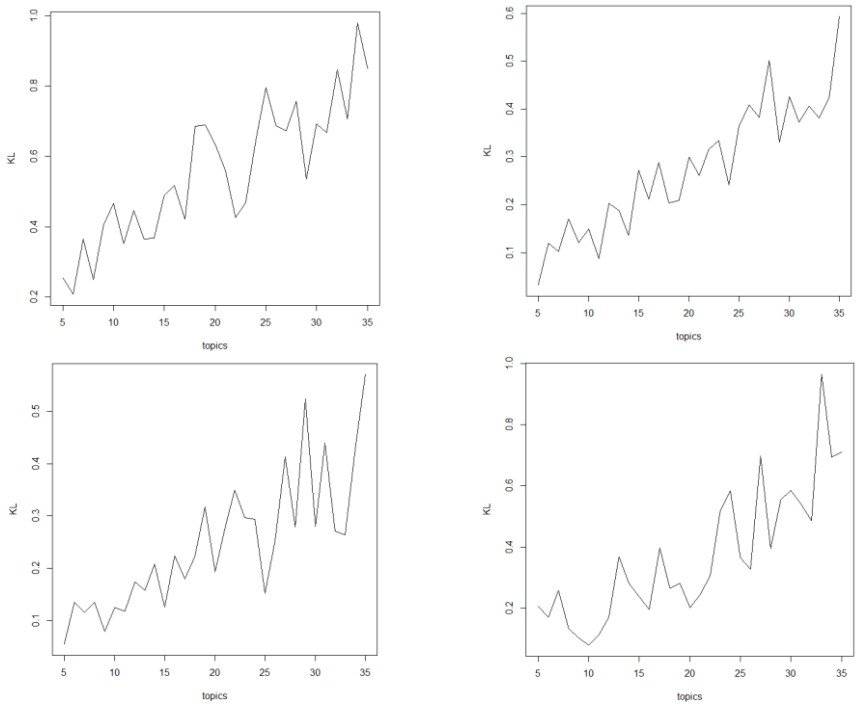

Figure 3. Variation of KL Divergence with Number of Topics

4. The Discovered Topics

We manually labeled the discovered topics. Table 3 shows selected topics and their top terms for the studied systems. Based on their semantic similarities, these groups seem to make sense to a human reader. It is very clear that the discovered topics are coherent and conceptually related. Any person with some familiarity with drawing tools, for instance, would concur that the terms 'g', 'diagram', and 'node' logically go together in this situation. We find the topics of the source code to be useful and coherent, just as different domains have found LDA topics to seem sensible and be suitable for their purpose.

Table 3. Example Topics and their Top Terms

System	Topic Name	Top Terms
ArgoUML	Figures	g, diagram, setting, node, edge, width, bound, param
	Files	le, project, name, pro le, label, name, panel, list
	Events	event, action, target, model, item, namespace, element, handle
Checkstyle	Formatting	check, indentation, type, level, param, child, parent, number
	Notifications	line, tag, warning, location, format, text, comment, annotation
	Rules	eld, support, pattern, rules, place, suite, test, missing
JHotDraw	Menu	view, editor, action, drawing, label, button, menu, chooser
	Figures	gure, handle, bound, event, link, area, point, connector
	Properties	color, icon, property, descriptor, component, index, inset, border
jEdit	Menu	event, action, handler, model, list, label, button, menu
	Search	node, count, search, history, property, result, pane, set
	Selection	line, selection, start, o set, bu er, text, color, area

The approach was capable of excerpting the domain topics, cross cutting topics, and even infrastructure-level topics. Table 3 also shows that the approach was capable of clustering all related keywords together. Another strong point in this approach is its

ability in resolving synonymy to an acceptable degree because LDA constructs topics in a le and words in a topic using multinomial probability distributions.

5. Threats to Validity

The approach of this paper depends on the quality of comments and identifier names found in the code. If developers of these systems did not follow any known naming conventions, this would results with low semantic value topics. However, the selected systems have a strict coding and naming conventions. In addition, a previous study revealed that most systems have good comments and good identifiers names which make them sufficient for such topic analyses [19].

We have focused on open source Java-based systems.We have selected these systems very carefully because of their extensive documentation, good designs, and manageable sizes. We tried varying the size of the selected system, however, we cannot generalize the results. Additional case studies are needed to investigate closed-source and other programming languages systems.

6. Related Work

Extracting topics from the unstructured text found in many software repositories has been the focus of many software engineering researchers. Most of their work is focused on calculating coupling/cohesion metrics, categorizing bug reports, and nding traces between requirements and code.

Thomas *et al.*, [6] evaluated topic modeling in analyzing the evolution of software using two well-documented systems. They computed different metrics on the discovered topic evolutions and found that most of topic evolutions matched with concrete code change activities by developers like corrective maintenance, improvements, and the addition of new features. Savage *et al.*, [11] developed a tool named 'TopicXP ', which extracts identifier names and comments from source code using Latent Dirichlet Allocation. The tool was implemented as open-source Eclipse plug-in that extracts and visualizes conceptual relationships between software entities. The goal of the tool is to support developers during software maintenance tasks. The tool helps developers to learn about the concepts, or latent topics, while also observing the dependencies and cohesiveness of these topics.

Gethers and Poshyvanyk [3] proposed a LDA-based coupling metric, the Relational Topic-based Coupling (RTC) metric. It is based on a variant of LDA called Relational Topic Models (RTM). RTM is an extension to LDA in which the links are explicitly modeled between documents in the corpus. These embedded links explains both the words of the documents and how they are connected. RTC utilizes these links to describe the coupling between two classes in the corpus. The authors demonstrated that the new metric is statistically di erent from existing metrics. Gethers *et al.*, [13] introduced an approach that uses Relational Topic Modeling (RTM) to recover traceability links between requirements and code. Their study showed a great promise in retrieving highly accurate tractability links. Oliveto *et al.*, [8] introduced an approach based on Relational Topic Models to identify Move Method refactoring opportunities and remove the Feature Envy bad smell from source code. Their approach analyzes both structural and semantic relationships between methods to identify sets of methods that share several responsibilities. Their empirical evaluation indicated that their approach provided meaningful refactoring opportunities.

7. Conclusion

Understanding the functionality of a large system is not an easy task. This paper investigated the applicability of LDA in extracting the main business topics and domains

from the source code. The goal was to discover the high-level functionality and the purpose of the system. One problem with unsupervised learning is to find the number of K, this paper adopt a well-known approach to finding the optimal K. Experiments on four Java-based open-source systems showed the effectiveness of the approach.

One direction of a future work is to investigate new techniques to excerpt topics at different granularity levels and classify different relationships between them. RTM moreover can be used in order to identify the relationships between method and classes. It can be even used to extract features interactions.

References

[1] X. Sun, X. Liu, J. Hu and J. Zhu, "Empirical studies on the nlp techniques for source code data preprocessing", Proceedings of the 2014 3rd International Workshop on Evidential Assessment of Software Technologies. ACM, **(2014)**, pp. 32-39.
[2] P. Anderson and M. Zarins, "The codesurfer software understanding platform", Proceedings. 13th International Workshop on Program Comprehension, IWPC 2005. IEEE, **(2005)**, pp. 147-148.
[3] M. Gethers and D. Poshyvanyk, Using relational topic models to capture coupling among classes in object-oriented software systems," in IEEE International Conference on Software Maintenance (ICSM), 2010. IEEE, 2010, pp. 1-10.
[4] M. Alenezi and K. Magel, Empirical evaluation of a new coupling metric: Combining structural and semantic coupling," International Journal of Computers and Applications, vol. 36, no. 1, 2014.
[5] D. M. Blei, A. Y. Ng, and M. I. Jordan, Latent dirichlet allocation," the Journal of Machine Learning Research, vol. 3, pp. 993-1022, 2003.
[6] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, Studying software evolution using topic models," Science of Computer Programming, vol. 80, pp. 457-479, 2014.
[7] T. Hofmann, Probabilistic latent semantic indexing," in Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 1999, pp. 50-57.
[8] R. Oliveto, M. Gethers, G. Bavota, D. Poshyvanyk, and A. De Lucia, "Identifying method friendships to remove the feature envy bad smell", Proceedings of the 33rd International Conference on Software Engineering. ACM, **(2011)**, pp. 820-823.
[9] K. Somasundaram and G. C. Murphy, "Automatic categorization of bug reports using latent dirichlet allocation", Proceedings of the 5th India Software Engineering Conference. ACM, **(2012)**, pp. 125-130.
[10] S. Wang, D. Lo, Z. Xing and L. Jiang, "Concern localization using information retrieval: An empirical study on linux kernel", 18th Working Conference on Reverse Engineering (WCRE), 2011. IEEE, **(2011)**, pp. 92-96.
[11] T. Savage, B. Dit, M. Gethers and D. Poshyvanyk, "Topic xp: Exploring topics in source code using latent dirichlet allocation", IEEE International Conference on Software Maintenance (ICSM), 2010. IEEE, **(2010)**, pp. 1-6.
[12] H. U. Asuncion, A. U. Asuncion and R. N. Taylor, "Software traceability with topic modeling", Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. ACM, **(2010)**, pp. 95-104.
[13] M. Gethers, R. Oliveto, D. Poshyvanyk and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery", 27th IEEE International Conference on Software Maintenance (ICSM), 2011. IEEE, **(2011)**, pp. 133-142.
[14] T. L. Gri ths and M. Steyvers, "Finding scientific topics", Proceedings of the National academy of Sciences of the United States of America, vol. 101, no. Suppl 1, **(2004)**, pp. 5228-5235.
[15] A. Kuhn, S. Ducasse and T. Girba, "Semantic clustering: Identifying topics in source code", Information and Software Technology, vol. 49, no. 3, **(2007)**, pp. 230-243.
[16] C. D. Manning, P. Raghavan and H. Schutze, "Introduction to information retrieval", Cambridge University Press Cambridge, vol. 1, **(2008)**.
[17] R. Arun, V. Suresh, C. V. Madhavan and M. N. Murthy, "On nding the natural number of topics with latent dirichlet allocation: Some observations", Advances in Knowledge Discovery and Data Mining. Springer, **(2010)**, pp. 391-402.
[18] T. M. Cover and J. A. Thomas, "Elements of information theory", John Wiley & Sons, **(2012)**.
[19] S. Haiduc and A. Marcus, On the use of domain terms in source code," in The 16th IEEE International Conference on Program Comprehension, ICPC 2008. IEEE, **(2008)**, pp. 113-122.

Author

Dr. Mamdouh Alenezi, received his Ph.D. degree in Software Engineering from Department of Computer Science at North Dakota State University, Fargo, ND in 2014. He got a Master's degree from DePaul University and got a Bachelor's degree from Prince Sultan University. His research interests include Mining Software Repositories, Software Maintenance, Software Testing, and Machine Learning.