Software Visualization Using Topic Models

Sandeep Reddivari
School of Computing
University of North Florida
Jacksonville, FL, USA 32224

William Hackney
School of Computing
University of North Florida
Jacksonville, FL, USA 32224

ABSTRACT

Latent Direchlet Allocation (LDA) is a statistical topic modeling approach that has been used to support several software engineering activities. The main assumption is that LDA offers a unique insight into the semantic content of software systems, thus revealing otherwise unseen relations between software artifacts. However, a main problem when dealing with LDA is the complexity of its output. In particular, the numerical probabilistic distributions produced by LDA to represent topics and documents are not intuitive to understand and rationalize. To address this problem, in this paper we present a topic modeling based approach to visualize software systems based on LDA. We also present several visualizations to represent the basic elements of LDA including words, topics, and documents. These different basic views are combined through a set of integration links to enable users to effectively explore software systems by supporting knowledge discovery at different levels of abstraction. We also demonstrate how the topic modeling based visualization approach can provide support to several software engineering activities such as program comprehension, software clustering, and code evolution analysis.

Keywords

Software visualization, program comprehension, topic modeling

1. INTRODUCTION

Software visualization can be defined as the mapping from software artifacts to graphical representations [16]. The main assumption is that using graphical representations of code structures reduces the cognitive effort required for understanding and navigating large and complex software systems [22]. However, extracting and visualizing meaningful information from source code is a non-trivial task [14, 18]. This can be explained based on the fact that the lexicons and syntax of programming languages is inherently more constrained than natural language [21]. Therefore, to be effectively visualized, the complex textual content of software systems has to be first reduced down to lower dimensional representations, while retaining as much of the original meaning of the text as possible [25]. These reduced representations can then be mapped into basic graphical objects to produce views of the system at higher levels of abstraction. A reductive transformation that has gained a considerable attention in Natural Language Processing (NLP) related tasks is Latent Dirichlet Allocation (LDA) [5]. Using LDA, the dimensionality of a large text corpus can be reduced down into a set of meaningful latent topics, where each topic consists of a group of words collectively representing a cohesive domain concept. In particular, LDA is an unsupervised probabilistic approach for estimating a topic distribution over a text corpus. The main assumption is that documents in the text collection are generated using a certain statistical generative model as random mixtures over latent topics. LDA has been successfully applied to several software engineering activities. However, a main problem when dealing with LDA is the complexity of its output. In particular, the numerical probabilistic distributions produced by LDA are not intuitive to understand and rationalize. This has motivated researchers to start looking for alternative ways to represent LDA's output [6, 13, 24].

Motivated by these observations, in this paper we propose a collection of visualization techniques, which are often used to represent topic models in NLP, to visualize software systems. These visualizations include combinations of basic and integrated views that facilitate effective navigation and comprehension of software systems. The rest of the paper is organized as follows. Section 2 briefly introduces LDA and our preprocessing analysis. Section 3 the various suggested visualization techniques of LDA. Section 4 presents visualization supports for clustering evaluation and code evolution analysis. Finally, Section 5 presents the conclusion and future work.

DOI reference number: 10.18293/SEKE2018-194

2. LATENT DIRICHLET ALLOCATION

LDA takes the documents collection D, the number of topics K, and α and β as inputs. Each document in the corpus is represented as a bag of words $d =< w_1, w_2, \ldots, w_n >$. Since these words are observed data, Bayesian probability can be used to invert the generative model and automatically learn ϕ values for each topic t_i, and θ values for each document d_i. In particular, using algorithms such as Gibbs sampling [23], an LDA model can be extracted. This model contains for each t the matrix $\phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$, representing the distribution of t over the set of words $< w_1, w_2, \ldots, w_n >$, and for each document d the matrix $\theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$, representing the distribution of d over the set of topics $< t_1, t_2, \ldots, t_n >$.

3. CODE VISUALIZATION USING LDA

This section describes the various visualization techniques used to represent the different aspects of the latent topic structure of software systems produced by LDA. We start

Table 1: Experimental Datasets

Dataset	VER.	NO. CLASS	LANG.	LOC	COMMENTS
iTrust	15.0	299	Java	20.7K	9.6K
Apache Ivy	2.3.0	451	Java	49.9K	16.7K
WDS	3.5.1	521	Java	44.6K	10.7K

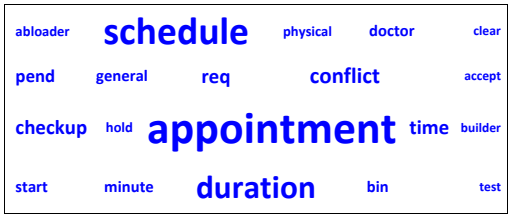

Figure 1: Tag cloud for a sample topic from the *iTrust* system

by describing basic views used for representing the basic units of LDA including words, topics, and documents. We then describe more sophisticated views where multiple basic views are integrated to represent the whole system at higher levels of abstraction.

3.1 Datasets

To start our analysis, we used three software systems from different application domains. Table 1 describes the characteristics of these systems including: the size of the system in terms of lines of source code (SLOC), lines of comments (CLOC), implementation language (LANG.) version (VER.) and number of classes (CLS).

3.2 The Topic View

This view emphasizes topics as the main unit of visualization. The topic view allows users to visually explore a topic as a collection of weighted words organized in a *tag cloud*. Tag clouds are visually-weighted renditions of collections of words (tags), extracted from a certain corpus, where more important words are depicted in a larger font size than less important words [17]. Importance can be quantified based on different schemes such as words counts, or the TFIDF weights of words. Tag clouds are widely used as an effective method to quickly find relevant information on the Web [3].

In our analysis, each topic is represented as a separate tag cloud. Each tag in the cloud represents a word from the topic-word distribution matrix. In particular, the size of the word in the cloud is proportional to its probability in the topic word matrix $\boldsymbol{P}(w_i, t)$ i.e,. words with higher probability are shown in larger font. Fig. 1 shows a tag cloud generated for a topic from the *iTrust* dataset. The topic is represented by 20 terms. The size of the different words <*appointment*, *schedule*, *duration*, ..., etc.> in the tag cloud in Fig. 1 shows that this particular topic describes the domain concept of scheduling a patient appointment.

3.3 The Document View

This view emphasizes artifacts as the main unit of visualization. In particular, each artifact (d) in the system is represented using a combination of graphical and textual components including:

- Topics chart: A standard pie chart which shows the topic distribution of each document. In particular, each sector of the pie represents a topic in the document-topic matrix. The size of the sector is proportional to the $\boldsymbol{P}(t_i, d)$ of that topic. A unique color is used to represent each topic [6].

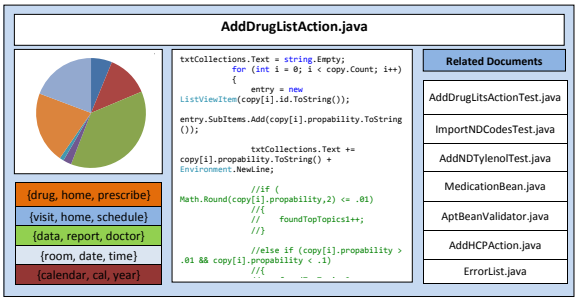

Figure 2: Document view of `AddDrugListAction.java`

- Topics list: A list of color-coded topics arranged in a descending order based on their $\boldsymbol{P}(t_i, d)$ value. The top three words of each topic are used as representatives of the topic; 3-5 terms were found to decently convey the main theme of the topic [7].

- Document text: A main window that shows the actual artifact (i.e., the actual source code of the class).

- Related artifacts: A list of other artifacts in the system ranked in a descending order based on their topical similarity to the main document. In particular, The similarity between documents d_1 and d_2 can be measured by the similarity between their corresponding topic distributions, using techniques such as the cosine similarity, or approaches such as the Kullback-Leibler (KL) divergence measure [26].

Fig. 2 shows a document view window for the class `AddDrugListAction.java` from the *iTrust* dataset. The view shows that the topic <*drug, home, prescribe*> is the dominant topic in this class. It also shows that the class `AddDrugListActionTest.java` is the most topically similar class to `AddDrugListAction.java`.

3.4 The Document-Topic View

This visualization provides an in-depth look into the topic distribution of each artifact in the system. In particular, stacked charts are used to represent the document-topic matrix of each document. Using stacked charts, the contribution of several data items into a total are represented as bars stacked one on top of, or next to, each other. In LDA, the width/height and color of each bar represents the $\boldsymbol{P}(t_i, d)$ of each topic representing the document d. Stacked charts are known to be effective and intuitive for comparing data distributions [11]. In our analysis, the main objective of this particular view is to enable the comparison of the documents at topic level. Comparing the topic distribution of documents is essential in several software engineering activities such as code evolution analysis [2] and traceability recovery [27].

3.5 The System-Topic View

This view is used to represent the whole system in a single view. We adopt distribution maps as the main visualization

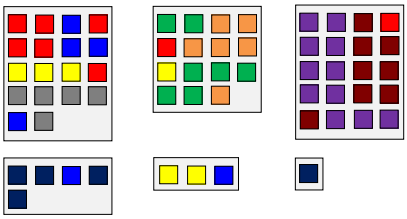

Figure 3: A topic distribution map of *Apachi Ivy*

for this view [10]. This visualization is composed of large rectangles containing small squares. Each rectangle represents a system folder or a package, and each square represents an artifact within that folder. The color of the square represents the dominant topic of the artifact. Distribution maps are suitable for showing the conceptual content of a corpus. In particular, they are good for visually representing metrics as such *focus*, or how well-encapsulated or cross-cutting a topic is, and *spread*, or the parts of the system in which this topic is present as a dominant topic [10].

Fig. 3 shows a partial distribution map for the *Apachy_Ivy* dataset. In particular, the map shows how some topics are spread over multiple folders, dominating multiple artifacts in each folder. It also shows how some other topics are focused (well-encapsulated) within one folder only.

3.6 The Integrated View

The integrated view combines several basic views to produce a full picture of the system, or one integrated visualization experience that allows users to navigate through the system's multiple views at different levels of abstraction. This view is enabled through a number of integration links that connect the different basic views of the system. Technically, such links could refer to a mouse click; in other implementations such as Web platforms, hyperlinks could be adopted. To integrate our views, several links have been added to the system. These links are shown in Fig 4. Such links include:

- In the system-topic views (i.e, distribution map), there is a link between each artifact's graphical object (square) and the document view of that artifact. An additional secondary link (right-click) is also available to show the tag cloud of the artifact's dominant topic.
- In the document view, a click on a topic in the list of document's topics lunches the tag cloud of the topic. Also, a click on any of the artifacts in the list of topically similar documents will lunch that artifact's document view.
- Links have also been added to the stacked charts views, where there is a link between each bar in the chart and the tag cloud of the topic represented by that bar.

The main objective of the integrated view is to facilitate effective program comprehension by enabling different comprehension strategies including top-down and bottom-up comprehension [19]. In particular, under the top-down strategy, software developers utilize their knowledge about the domain to build a set of expectations that are mapped onto the source code [19]. The top-down comprehension process starts from the distribution map view of the system, where developers who are usually familiar with the domain,

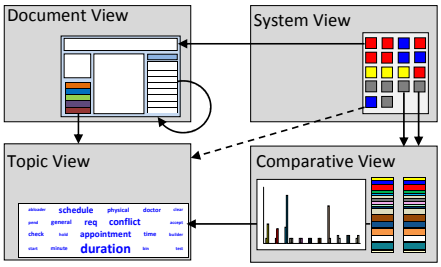

Figure 4: Integrated View

can explore the system as one unit. Such knowledge can then be propagated through the integration links down to the basic views of individual artifacts and topics, where users can investigate the source code of individual classes in the document view and the word distribution of each topic through the topic's tag cloud.

On the other hand, a bottom-up comprehension approach adopts a divide and conquer strategy. In particular, the comprehension process starts from the source code, where developers who are often unfamiliar with the domain can then gradually build a full understanding of the system. This process is also enabled by moving to higher abstraction levels through the integration links, starting from basic document's and topic's views, up to the system's distribution map to produce a full understanding of the system from the bottom-up.

4. VISUALIZATION SUPPORT

4.1 Clustering Evaluation

LDA has been intensively used as a mechanism for clustering software systems. In particular, the topic distributions of artifacts are used as main features to group topically related artifacts into smaller, more conceptually cohesive, and thus, easier to understand subsystems [12, 21]. However, finding best LDA clustering settings (e.g., number of topics, α, ϕ, clustering algorithm to use, number of clusters, and the distance function) that best fit a certain task is often described as an NP-complete problem [28].

Several internal and external methods have been proposed in the literature to estimate optimal clustering parameters. Internal methods use a fitness function that captures the twin objectives of high cohesion and low coupling to help determine the boundaries between clusters [8]. On the other hand, external measures (e.g., MoJo [29]) use an authoritative decomposition as a reference point to assess the quality of generated clusters. However, such methods reduce the overall evaluation into a single number, hiding valuable information about the nature of the problem. To that end, visualizing clustering results can help to assimilate such information, providing insights into the operation of the different clustering algorithms, and their sensitivity to different clustering parameters, such as number of clusters, membership, and boundaries. [9]

In the following example, we demonstrate how topic modeling based visualizations can be used to visually compare clustering algorithms. In particular, we experiment with Complete Linkage (CL) and Single Linkage (SL), two Hierarchical Agglomerative Clustering (HAC) algorithms that have been showing consistent performance in several soft-

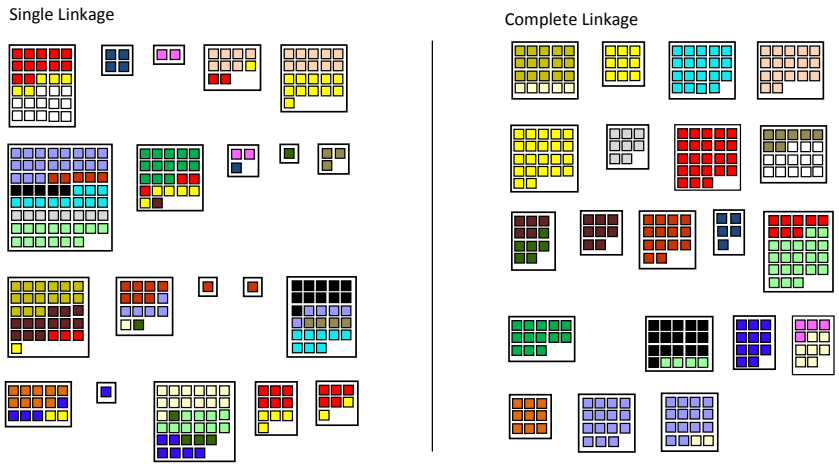

Figure 5: Visually comparing the clustering results of SL and CL algorithms on *iTrust*

ware engineering activities [1]. Technically, both SL and CL start from individual items in the cluster space, where each item is treated as a separate cluster, iteratively merging items based on a certain update rule, until the whole system is grouped into a single cluster. This process produces a hierarchy of clusters, or a dendrograph, of the whole process. Formally, SL and CL can be defined as follows:

$$SL = min\{d(a,b) : a \in A, b \in B\}. \quad (1)$$

$$CL = max\{d(a,b) : a \in A, b \in B\}. \quad (2)$$

In these equations, $d(a,b)$ is the distance between data objects a and b, defining the linkage (merging) criteria for clusters A and B. For instance, SL merges the two clusters with the smallest minimum pairwise distance, and CL merges the two clusters with the smallest maximum pairwise distance. Cosine similarity is used to measure the topical distance between the different artifacts.

We apply LDA using K = 20 to our *iTrust* dataset. We then cluster the system's artifacts using both CL and SL algorithms. We set both algorithms to produce 20 clusters. We then use MoJoFM to assess the performance of both algorithms by comparing their output to the optimal decomposition [29]. An optimal decomposition is the one that groups documents that match in their dominant topics into separate clusters. Since we produced 20 topics for the system, the optimal decomposition consists of 20 clusters.

MoJo measures the distance between two decompositions of a software system by computing the number of Move and Join operations to transform one to the other. MoJoFM is basically a normalized MoJo that produces a number in the interval [0, 1]. MoJoFM value of 1 means that the algorithm is optimal. Applying MoJoFM on our clustering output returns values of $< 87.71\%, 43.1\% >$ for CL and SL respectively, giving a clear indication of the superiority of CL over SL; however, no other information is provided. To reveal such information, we refer to our distribution maps to visually compare the output of the two algorithms. Results are shown in Fig. 5. Each map shows how each clustering algorithm divided the 299 artifacts of the *iTrust* system among the 20 clusters.

Fig. 5 shows that SL produced unbalanced clusters, scattering some of the topically related artifacts all over the clustering space. This behavior of SL can be explained based on its update rule, SL uses the smallest minimum pairwise distance between cluster items as the new distance in the newly formed clusters. Therefore, when SL is used, bigger clusters tend to be grouped together rather than incorporating singletons (clusters with only one element). As a result, SL tends to create a small number of large, isolated clusters, in addition to a number of singletons. In constant, CL uses the smallest pairwise maximum distance to merge clusters, thus pushes clusters apart, creating smaller, more balanced, and highly cohesive clusters.

Table 2: Clustering Results

Algorithm	MoJo	Avg. Spread	Avg. focus
Complete Linkage	87.71%	1.35	82%
Single Linkage	43.1%	2.35	48%

To confirm our visual assessment, we refer to the *focus* and *spread* metrics associated with distribution maps. Such values, averaged over all the topics in *iTrust*, are shown in Table 2. The *spread* values show that using SL, artifacts sharing same dominant topic are spread over an average of 2.35 clusters, while in CL, the average spread is 1.35, giving an indication of the high encapsulation of topically related artifacts in CL clusters. Similarly, we calculate the focus values of both algorithms using the following formula:

$$focus(t,P) = \sum touch(t,p_i) * touch(p_i,t) \quad (3)$$

where $touch(t,p_i)$ is the ratio of the number of artifacts in p_i with dominant topic t to the total number of artifacts in p_i, and $touch(p_i,t)$ is the ratio of the number of artifacts in p_i with dominant topic t to the total number of artifacts in the system with a dominant topic t [10]. Focus values show that SL was less successful in producing cohesive clusters ($focus = 48\%$), producing larger clusters that contain multiple sets of topically related artifacts. In contrast, CL produced more cross-cutting clusters ($focus = 82\%$).

In general, using our distribution map view, it can be visually concluded that CL was more effective than SL in dividing the system's artifacts into more topically cohesive and more balanced clusters. In particular, visualization helps users to visualize the different metrics and the meaning of these numbers, providing an effective alternative for illustrating the impact of the various clustering parameters, and facilitating a real-time comparison of different clustering re-

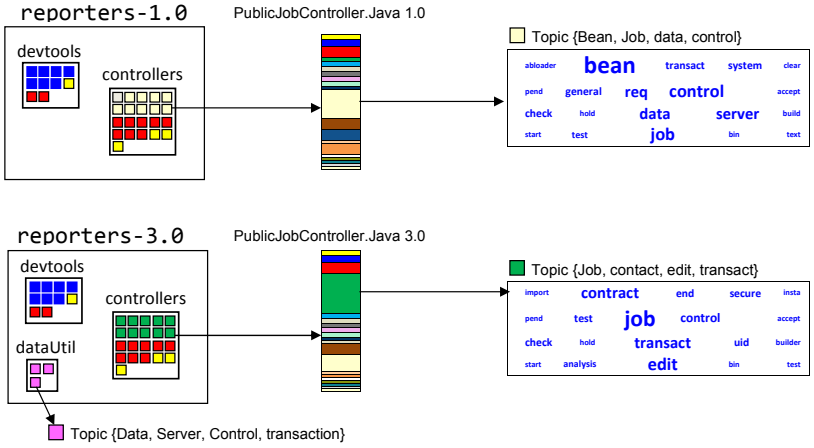

Figure 6: Visually analyzing the change in one of the *WDS* system's folders

sults. For instance, the concentration of colors inside boxes gives an indication of high encapsulation and clear cross-cutting. In addition using the integrating links, the user can get a better understanding of the results, for example why certain artifacts with different dominant topics tend to be grouped together.

4.2 Code Evolution Analysis

Several studies have reported that topic models can be effectively used for the purposes of describing software evolution [20]. The main assumption is that the changes in the latent topic structure of a software system are reflective of actual changes made by developers, for example feature integration or refactoring milestones [27]. In particular, such events can be detected by applying LDA to different releases of the system and monitoring the changes in a system's topic structure over time.

For instance, the sensitivity of LDA to software evolution can be quantified through matching individual artifacts based on their document-topic distribution, or comparing how the topic-word matrices of the different topics in the system have changed over time using well-defined evolution metrics [27]. However, similar to software clustering, such analysis can reveal only a little about the nature of the change [15]. For instance, other evolution-related information such as what specific types of change took place, and which artifacts have been affected, is often invisible to such metrics. To that end, visualization can help developers and software engineers to uncover such information by providing a more in-depth look into the different releases of the system overtime. In particular, to visualize a software change, a time dimension is integrated into the different views of the software system [15]. For example, to visualize system artifacts' evolution, we add a time dimension to our document-topic view. Similarly, a time dimension is integrated into the system-topic view of the system to visualize changes in the system as a whole.

To demonstrate this process we run LDA over two releases of the *WDS* dataset. In particular, we work with releases 1.0 and 3.0 of the system as milestone changes have been reported between these two particular releases. In order to keep a consistent color assignment, if two topics generated for different releases of the system share the top five dominant words, then the same color is used. Using our zooming feature, we produce distribution maps of one the system's sub-folders (`reporting`) in both releases of the system. In release 1.0, this particular folder contains two sub folders including `controllers` and `devtools`. In release 3.0 the same folder now includes the additional folder `dataUtilities`.

Distribution maps, shown in Fig. 6, show that artifacts in folder `devtools` remain unchanged between releases. However, a drastic change in a portion of the artifacts in folder `controllers` has happened. To understand this change we take a look at the stacked chart of the system class `PublicJobController.Java` in both releases. The charts show that in release 1.0 the topic $<bean, job, data, control>$ was dominating this class. This topic describes functionalities related to job requirements and database connections. However, in release 3.0 this class is now dominated by the topic $< job, contact, edit, transact >$, and the data related terms are no longer present. In an attempt to understand this change we inspect the new folder `dataUtlities` that has emerged in the folder `reporting` in release 3.0. All artifacts in this folder are dominated by the same topic. It can be inferred from the word distribution of this topic $< data, server, control, transaction >$ that this folder basically contains database related functionalists. This suggests that probably Extract Class [4] refactoring has taken place somewhere between releases 1.0 and 3.0, where most of the database related features have been moved and encapsulated into the new folder `dataUtlities`, leaving artifacts in folder `controllers` with only job-controlling related functions.

This example demonstrates how a change in the system was made immediately obvious by our views, allowing users to not only identify the change, but also provide insight into the nature of the change.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a set of visualization techniques to represent source code using topic models. The main assumption is that using graphical representations to represent complex source code structures helps to reduce the cognitive load when comprehending a software system. In particular, we used LDA as an effective dimentionality reduction technique to reduce the inherently complex textual content of source code into a set of semantically cohesive topics that can be effectively visualized. Such topics are represented through several views that have been used in a wide range of NLP applications. These views provide graphical representations for the different numerical distributions produced by LDA including words, topics, and documents. These dif-

ferent basic views are integrated through a set of links to enable users to quickly browse through the system modules, exploring relationships between artifacts that might otherwise go unnoticed. In addition, we presented how the topic modeling based visualization can provide support to software engineering scenarios such as program comprehension, code clustering, and evolution analysis. In the future we plan to implement our proposed visualizations through a working prototype which provides several options to adjust the visualization settings and to navigate through the different views of the system.

6. REFERENCES

[1] N. Anquetil, C. Fourrier, and T. Lethbridge. Experiments with clustering as a software remodularization method. In *Working Conference on Reverse Engineering*, pages 235–255, 1999.

[2] H. Asuncion, A. Asuncion, and R. Taylor. Software traceability with topic modeling. In *International Conference on Software Engineering*, pages 95–104, 2010.

[3] S. Bateman, C. Gutwin, and M. Nacenta. Seeing things in the clouds: The effect of visual features on tag cloud selections. In *ACM Conference on Hypertext and Hypermedia*, pages 193–202, 2008.

[4] G. Bavota, A. D. Lucia, and R. Oliveto. Identifying extract class refactoring opportunities using structural and semantic cohesion measures. *Journal of Systems and Software*, 84(3):397–414, 2011.

[5] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[6] A. Chaney and D. Blei. Visualizing topic models. In *AAAI Conference on Social Media and Weblogs*, 2012.

[7] J. Chang, J. Boyd-Graber, S. Gerrish, C. Wang, and D. Blei. *Reading tea leaves: How humans interpret topic models*, pages 288–296. Curran Associates, 2009.

[8] J. Davey and E. Burd. Evaluating the suitability of data clustering for software remodularization. In *Working Conference on Reverse Engineering*, pages 268–277, 2000.

[9] I. Davidson. Visualizing clustering results. In *SIAM International Conference on Data Mining*, 2002.

[10] S. Ducasse, T. Gírba, and A. Kuhn. Distribution map. In *International Conference on Software Maintenance*, pages 203–212, 2006.

[11] S. Eick. Visualizing software changes. *IEEE Transactions on Software Engineering*, 28(4):396–412, 2002.

[12] S. Grant and J. Cordy. Estimating the optimal number of latent concepts in source code analysis. In *International Working Conference on Source Code Analysis and Manipulation*, pages 65–74, 2010.

[13] B. Gretarsson, J. OĎonovan, S. Bostandjiev, T. Höllerer, A. Asuncion, D. Newman, and P. Smyth. Topicnets: Visual analysis of large text corpora with topic modeling. *Journal of Visual Languages and Computing*, 3(2):2157–6904, 2012.

[14] M. Hearst. Tilebars: Visualization of term distribution information in full text information access. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 59–66, 1995.

[15] R. Holt and J. Pak. GASE: Visualizing software evolution-in-the-large. In *Working Conference on Reverse Engineering*, pages 163–167, 1996.

[16] R. Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey. *Journal of Software Maintenance*, 15(2):87–109, 2003.

[17] B. Kuo, T. Hentrich, B. Good, and M. Wilkinson. Tag clouds for summarizing web search results. In *International Conference on World Wide Web*, pages 1203–1204, 2007.

[18] R. Laramee. Using visualization to debug visualization software. *IEEE Computer Graphics and Applications*, 30(6):67–73, 2003.

[19] S. Letovsky. Cognitive processes in program comprehension. In *workshop on empirical studies of programmers on Empirical studies of programmers*, pages 58–79, 2011.

[20] E. Linstead, C. Lopes, and P. Baldi. An application of Latent Dirichlet Allocation to analyzing software evolution. In *International Conference on Machine Learning and Applications*, pages 813–818, 2008.

[21] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining concepts from code with probabilistic topic models. In *International Conference on Automated Software Engineering*, pages 461–464, 2007.

[22] J. Maletic, A. Marcus, and M. Collard. A task oriented view of software visualization. In *International Workshop on Visualizing Software for Understanding and Analysis*, pages 32–40, 2002.

[23] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling. Fast collapsed gibbs sampling for Latent Dirichlet Allocation. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 569–577, 2008.

[24] T. Savage, B. Dit, M. Gethers, and D. Poshyvanyk. TopicXP: Exploring topics in source code using latent dirichlet allocation. In *IEEE International Conference on Software Maintenance*, pages 1–6, 2010.

[25] K. Sparck-Jones. Automatic summarising: Factors and directions. In *Advances in Automatic Text Summarization*, pages 1–12, 1998.

[26] M. Steyvers and T. Griffiths. *Probabilistic topic models*, pages 427–448. Psychology Press, 2007.

[27] S. Thomas, B. Adams, A. Hassan, and D. Blostein. Validating the use of topic models for software evolution. In *IEEE Working Conference on Source Code Analysis and Manipulation*, pages 55–64, 2010.

[28] K. Tian, M. Revelle, and D. Poshyvanyk. Using Latent Dirichlet Allocation for automatic categorization of software. In *International Working Conference on Mining Software Repositories*, pages 163–166, 2009.

[29] Z. Wen and V. Tzerpos. An effectiveness measure for software clustering algorithms. In *International Workshop on Program Comprehension*, pages 194–203, 2004.