Interactive Visualization of Bug Reports using Topic Evolution and Extractive Summaries

Shamima Yeasmin Chanchal K. Roy Kevin A. Schneider
Department of Computer Science, University of Saskatchewan, Canada
shy942@mail.usask.ca, {chanchal.roy, kevin.schneider}@usask.ca

***Abstract*—Software bug reports are important project artifacts that evolve throughout the life of a software project. Software bugs are issues that are reported by users when these issues hinder their work. Software projects evolve over time as bugs are addressed and new features are added. Managing bugs can be a significant challenge as a project manager generally needs to be aware of all the bug reports for the current version, and this can be even more challenging when the number of bug reports becomes large. It is preferable that a developer new to a project improves her knowledge with the project along with the bug reports during working on it, which is likely to help her avoid or handle the reported issues. In this paper, we propose a prototype that assists developers review a project's bug reports by interactively visualizing insightful information regarding the bug reports using topic analysis. In addition, in order to reduce developers' time and efforts when studying a bug report, the proposed prototype also provides an extractive summary visualization of each bug report. In this research, it is shown that our proposed prototype performs better in terms of precision, recall, and F-measure than a baseline approach that uses time-sensitive keyword extraction.**

Keywords*—Bug report, Topic evolution, Summarization, Interactive visualization

I. Introduction

Software bug reports are an important source of information for software development and maintenance. During resource allocation, a project manager distributes time and effort for each project task and assigns appropriate developers. Estimating resource requirement is a time-consuming task that also involves studying a large number of previously filed bug reports. The idea is to determine which part of a given project is more problematic and thus needs more attention. The task is trivial when there are only a few bugs reported. However, the number of bugs typically increases over time, and it becomes almost impossible for a manager to analyze such a huge collection of bug reports. When a developer starts working on an existing project, it is preferable that she first strengthens her knowledge with the project along with the bug reports, which also costs development time and effort. Thus, analyzing a number of bug reports manually is highly unproductive, and an automated tool support is likely to benefit in this regard. In this paper, we propose a prototype that provides insightful information derived from a collection of bug reports as well as visualizes the extractive summary of a bug report.

In our research, we perform two major visualization tasks on software bug reports. The first shows topic evolution over time derived from a collection of bug reports. However, We applies topic modeling on a collection of bug reports and produces a visual topic-based analysis. Topic modeling is a machine learning technique that provides information regarding the hidden structure of a collection of documents [1]. Topics are constructed statistically by identifying co-occurring words and summarizing key concepts of a document collection. Topic modeling on bug reports instantly provides an overview about the different parts of a project containing significant number of bugs, and the project managers can take effective steps in resource allocation. By inspecting topic evolution over time in a time-windowed manner, a developer can be aware of the frequent bugs occurred in the past.

The second task visualizes the extractive summary of a bug report when requested by a developer. We create the extractive summary of a bug report using the hurried bug summarization technique proposed by Czarnecki et al. [2]. However, Extractive summaries are often hard to understand as the sentences are taken out of their context in the bug reports, and thus the sentences may lose their consistency. In this situation, if a summary sentence can be revealed and visualized in the bug report itself with different colour coding, the developer can further investigate the context (i.e., the sentences that precede or follow the summary sentence), which influences the meaning of the sentence. We provide an interactive visualization of a bug report summary, to aid a developer view the original bug report context. Thus, a project manager or a developer can study a bug report more thoroughly but with less effort. In this paper, our contributions include: (i) A topic evolution visualization for bug reports, which is a convenient way to help developers rather than using only textual information. (ii) A detailed drill-down from a topic's time-segments to its related software bug reports, (iii) A searching facility to help developers explore related issues regarding a given topic-keyword, and (iv) An interactive visualization of a bug report summary that conveniently links summary sentences to their context. We collect 3914 bug reports from Eclipse Ant, and analyze them with our system. We also evaluate the quality of time-segment keywords of top five topics and it is shown that our system outperforms term frequency based system in case of precision, recall, and F-measure.

II. Related Work

To represent the evolution of bugs D'Ambros et al. [4] propose a visualization technique called **System Radiography** indicating which are the most problematic parts of the system. They also provide useful information regarding the life cycle of a bug by another visualization technique, **Bug Watch**. A different visualization technique is proposed by Dal Sassc and

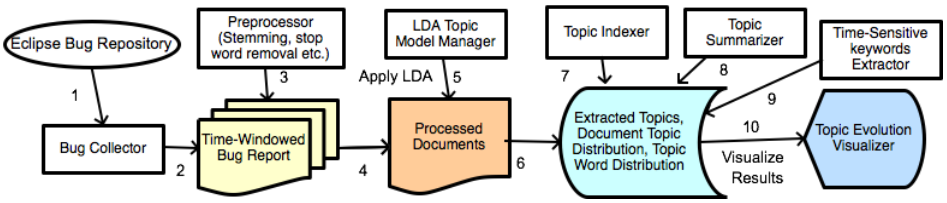

Fig. 1: Schematic Digram of Topic Evolution Visualizer

Lanza [3] to represent a fine-grained view of a bug report. To analyze bug tracking system, they also propose a visual analytic platform, called **in*Bug**. Hora et al. [6] present a tool, **BugMaps** to map reported bugs to defects in the classes of object oriented systems and provide many interactive visualizations for decision support. The main differences between those work and our prototype are that (i) for visualization of bugs, we are using topic evolution over time, but D'Ambros et al. [4] use matrix-based representation, Dal Sassc and Lanza [3] propose a web-based visual analytics platform, and Hora et al. [6] utilize Distribution Map and (ii) that none of those work visualizes bug report extractive summaries which we do.

Martie et al. [7] apply LDA topic modeling on a large document consisting of Android developers' discussion data to analyze the development of the Android open source project. A limitation of this work is that although they consider discussion trends over time they use the same associated keywords throughout. Topic evolution seems to be meaningless to developers if a topic's associated keywords do not change over time. To mitigate this problem in our work, for each time-window we extract frequently used keywords associated with each topic, which we refer as time-sensitive keywords. Thus, our approach is more time specific than theirs.

In the field of information visualization, Havre et al. [5] use a symmetric river metaphor to represent thematic variations over time in the context of a time-line and corresponding external events. TIARA (Text Insight via Automated Responsive Analytics) [8] conveys far more complex text analysis results than Havre et al. [5] by showing detailed thematic content in keywords. TIARA is a visual analytic system that shows the content evolution of topics over time [8]. They extract topics from email data and patient records, and generate time-sensitive keywords to represent topic evolution. The differences between their tool and our prototype is that (i) we are demonstrating topic evolution of software bug reports rather than email data with some variations in techniques and (ii) we are also visualizing extractive summaries which they did not.

In this research, besides showing topic evolution of bug reports, we also applied the hurried bug summarization approach proposed by Czarnecki et al. [2] to create summaries of bug reports and then visualize them in a convenient way to increase the understandability for the developer. To best of our knowledge, no visualization has yet been done for bug report extractive summaries. Thus, we are the first proposing this kind of visualization.

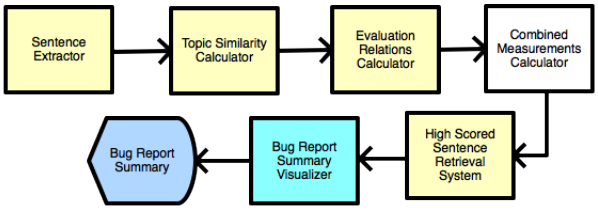

Fig. 2: Schematic Diagram of Bug Report Summarizer

III. Proposed Approach

We propose a standalone prototype, which requires the repository of bug-reports downloaded from official bug-repositories such as Bugzilla[1]. We divide our proposed prototype in two different parts: Part I is related to the features, which are based on topic modeling and Part II creates extractive summaries of bug reports and visualizes them.

We further divide the topic evolution part (Part I) into two individual phases: analytics and visualization. In the analytics phase, we collect bug reports from a popular bug tracking system, perform preprocessing such as stemming, stop word removal and so on, then we apply LDA topic modeling on the preprocessed data utilizing Gibbs Sampling to extract topics. We utilize JGibbLDA[2], which is a Java implementation of LDA, to apply topic modeling on our dataset. Then we filter keywords per topic and finally extract time-sensitive keywords for each time interval. In the visualization phase, we visualize topic evolution with the help of a popular Java chart library JFreeChart[3], where the X-axis represents time and the Y-axis represents the number of bugs containing that topic. We implement our prototype in Java and perform experiments using Mac OS X 10.8. Our proposed system architecture for part I is shown in Fig. 1.

In the summary visualization part (Part II) shown in Fig. 2, we create bug report summaries utilizing methods described in Czarnecki et al. [2]. Cosine similarity is a measure to determine the lexical similarity of two sentences. In our system, we calculate cosine similarity of each sentence with all other sentences and with the title of the bug report. If a sentence is evaluated by any other sentence, then a directed link is created among them, which is referred to as the evaluation relationship. We utilize twitter sentiment analysis API[4] to determine the

[1]https://bugs.eclipse.org/bugs/
[2]http://jgibblda.sourceforge.net
[3]http://www.jfree.org/jfreechart/
[4]http://help.sentiment140.com/api

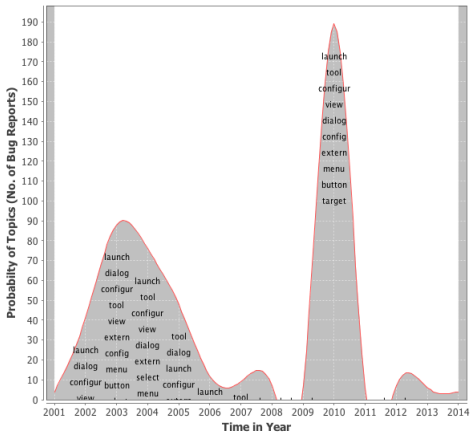

Fig. 3: Topic Evolution Example (Topic 3: Tool Launch and Configuration)

sentiment of each sentence and then compute the evaluation relationship score for each sentence and finally combined them to rank the sentences of a given bug report [2]. Furthermore, we visualize the bug report summary using different colour combinations to indicate the summary sentences from the context to aid the developer.

IV. Proposed Visual Designs

One of the main objectives of our proposed prototype is to provide insightful information to developers through software bug reports as they evolve over time. That is why, we design our prototype so as to provide the highest possible information interactively. Currently, our visualization prototype: (i) generates as well as shows topic evolution of each topic automatically, (ii) then for further inspection it retrieves all software bug reports associated with a given topic along with their Bug Report IDs and titles, (iii) provides a searching option so that a developer can search bug reports by keywords associated with a topic, and (iv) visualizes an extractive summary of each bug report. We utilize an area-graph based visual layout to represent topic evolution (i.e., content changes over time as in Fig. 3). Our prototype generates the visual summarization of each topic individually. This area layout is depicted by set of keywords clouds in order to show the content evolution over time. The height of the area graph at each time-segment (here, a year) encodes the strength of the topic for that point (Fig. 3). Strength is calculated by the number of software bug reports containing that topic at the certain point. The functionality of our visualization prototype is described in the following subsections.

***A.* Topic Evolution of a Collection of Bug Reports:** Once a developer selects a dataset (i.e., a collection of bug reports) the system automatically applies topic modeling to it. After performing some analytics on the produced topic model, the prototype depicts the topic evolution of several topics derived from the dataset. From this visualized output as in Fig. 3 and Fig. 7, developers can analyze which type of topics are evolved most of the time and associated with most of the bugs.

***B.* Drilldown Inspection in Context:** In LDA topic modeling, keywords that are assigned to a topic are extracted automatically. Therefore, two situations may occur: (i) there might be one or more associated keywords are not important

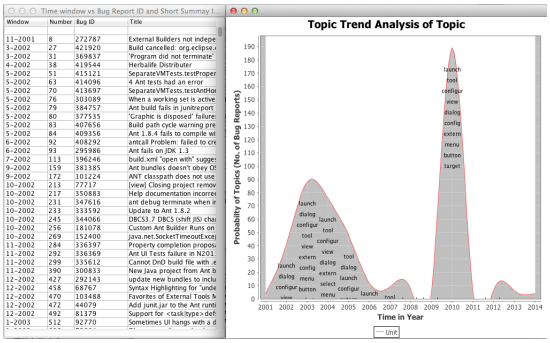

Fig. 4: Topic Drill Down in the Context

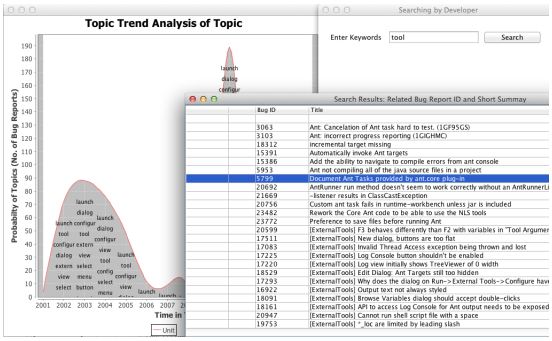

Fig. 5: Search by Keywords

enough to understand the topic, and (ii) the developer may not understand the topic-keywords relationships. Therefore, in these circumstances to help the developer to understand a topic, our prototype provides the opportunity to drilldown from the topics to the document collection level. The only way to gain more knowledge about the topic-keywords relationships is by studying the context in the document collection, from which they have come. So, if the developer requests for more information regarding a topic, then all bug reports IDs and titles associated with that topic will be shown, as is depicted in Fig. 4. In this way the context of the bug reports will aid the developer in gathering enough knowledge to identify that topic precisely.

***C.* Searching by keywords** The searching facility is presented in Fig. 5. A topic is associated with several keywords. A developer may be interested in inspecting any of them. Consider a scenario where a developer finds a topic described by keywords such as 'launch', 'tool', 'view' and so on as in Fig. 5. She may be curious to know which type of 'tool' related bugs are mentioned by this topic. To help her we provide a search option, where the developer is able to perform a search on the entire bug report collection for a keyword. In Fig. 5, a search result is shown containing bug report IDs and titles for the keyword 'tool'.

***D.* Summary Visualization** The visualized bug report summary is presented in Fig. 6. During designing the visual summary of a bug report, we kept two things in mind: first, the length of the summary should be significantly smaller than the original bug report so that the developer will have fewer sentences to study; and second, the visualization must be done in a way that can fulfill a developer's intention for reading it properly. Therefore, to create a good summary we follow the approach proposed by Czarnecki et al. [2] which is automatic

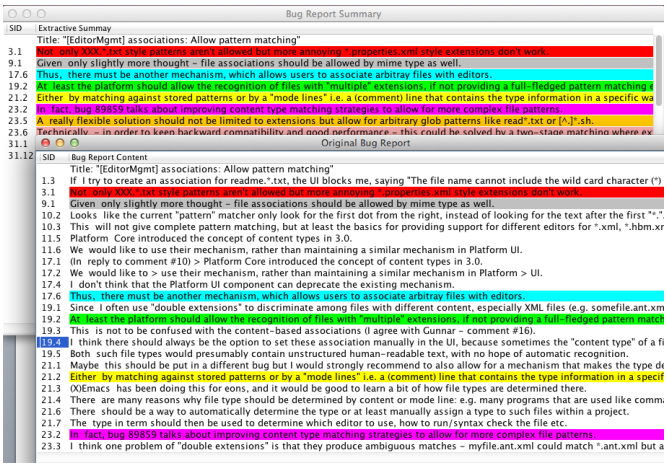

Fig. 6: Bug Report Summary Visualization

and extractive. We also restrict the number of sentences of our summary to ten. In our proposed prototype, the visualized summary is represented along with the visualized original bug report to the developer. Each sentence in the summary is coloured with a unique colour and the same sentence in the original bug report is coloured by the same colour. This kind of visualization can help the developer to understand the summary from the context. As the summary we have created is extractive, sometimes it may be difficult for the developer to gather the desired idea from it. That's why, when the summary sentences are also highlighted in the original bug report utilizing the same colours, the developer is able to study the sentences that precede and follow the summary sentences in original bug report, which can aid her in understanding summary sentences in context.

V. An Example Case Study

To examine the effectiveness of topic evolution, let us assume a scenario where a novice developer has started working on Eclipse-Ant, so at the beginning she wants to give a quick look at its bugs. At present, Eclipse-Ant contains 3914 bug reports and definitely studying all of them would take a long time. Therefore, in this situation we are providing our prototype that can aid her to more conveniently and quickly dig deeper into a large collection of bug reports including the bug reports from the prevision versions. From our prototype we can see 20 topics as output, each of which have 10 keywords. To keep this discussion simple, an example of the top most five topics together with their associated keywords are provided in Table I. We see that the 1st, 2nd and 3rd topics are about 'Plug in', 'Editor' and 'Tool' respectively. Furthermore, from the visualization of a topic such as in Figure 3, she can explore when a topic peaked. However, If we search with these keywords both in Bugzilla and our prototype; Bugzilla returns less bug reports compared to our proposed prototype for the same queries as shown in Table II. To investigate the reason behind this, we randomly as well as manually check results both from our proposed prototype and Bugzilla. Here, in Bugzilla almost all retrieved bug reports contain searching keyword in their titles, because Bugzilla produces search results based on bug report titles only, whereas we consider the contents of the bug report in addition to the title during searching.

TABLE I: An Example Topic with Keywords and Labels

No.	Topic Label	Keywords
1	Plugin Support	require user issu possible feature support realli gener plugin plan
2	Editor Outline	editor xml view outlin content action elem open docum associ
3	Tool Launch and Configuration	launch tool dialog configur view extern config select menu button
4	Version log	reproduce memori version view instal window attach open log time
5	Page Preference	tab page prefer buttoon classpath dialog home runti default pref

TABLE II: Search Results in terms # of bug reports retrieved

Keyword	Bugzilla	Proposed Tool
plugin	58	577
editor	371	702
tool	469	860
log	191	518
tab	130	533

TABLE III: # of Bug Reports in Eclipse-Ant from 2001 to 2014

Year	# Bug Reports	Year	# Bug Reports
2001	17	2008	114
2002	484	2009	510
2003	776	2010	90
2004	728	2011	128
2005	479	2012	71
2006	256	2013	91
2007	155	2014	15

In our scenario, from Table I, the novice developer can gather an idea regarding the most occurring problems (i.e., bugs) in Eclipse-Ant, which are related to 'plug-in', 'editor', 'tool', 'log' and so on. To dig deeper she can also search by those keywords as in Fig. 5, and can have a clear idea about how many bug reports are associated with each top topic. Below are some questions we can use our prototype to address for the above scenario: (i) Which month/year was the most crucial period for Eclipse-Ant bugs?; (ii) What was the most active topic in a given year such as 2003? and (iii) What was discussed in the most active topic? and so on.

To answer the first question we need to investigate the top most topics, where they are in their peak. Here, we can see that the year 2009 is the most active year for Eclipse-Ant bug reports for the top most five topics, two of them, topic-3 and topic-4, are depicted in Fig. 3 and Fig. 7 respectively. Then 2003 is the second most active year for this bug dataset. We also can relate it to the number of bug reports of Eclipse-Ant in each year from 2001 to 2014 as presented in Table III. Here, although years 2003 and 2004 have the highest number of bug reports, the top-most five topics are not that active in these two year, compared with year 2009. To address the second question, we notice that in 2003 the top five topics (Table I) have the following number of bug reports, respectively: 63, 11, 88, 26, and 25. That means topic-3, "Tool Launch and Configuration", is the most active topic during 2003. It is also observed from Table I that the most crucial topic, i.e., topic-3 contains keywords such as 'launch, 'tool', 'dialog', 'configur', 'view' and so on. We can verify it with searching results presented in Table II, where we can see that the highest number of bug reports are retrieved against keyword 'tool' both from Bugzilla and our proposed tool. If requires she is able to

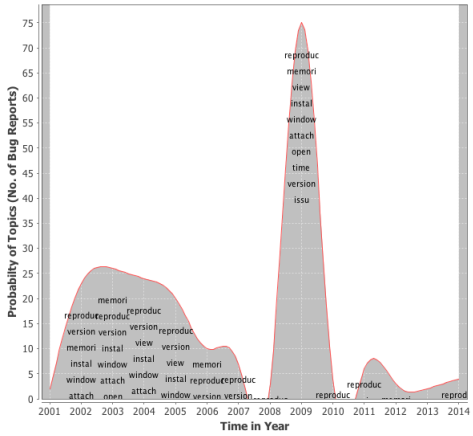

Fig. 7: Topic-4 (Version log)

TABLE IV: Precision, Recall and F-measure for top 5 Topics

Topic No.	TF-Based System			Proposed Tool		
	Precision	Recall	F-measure	Precision	Recall	F-measure
1	0.1613	0.2	0.1786	0.44	0.66	0.5280
2	0.1711	0.26	0.2063	0.5231	0.68	0.5913
3	0.1228	0.14	0.1308	0.5263	0.60	0.5607
4	0.2154	0.28	0.2435	0.5555	0.60	0.5769
5	0.2308	0.3	0.2609	0.5926	0.64	0.6154
Mean	**0.1803**	**0.2360**	**0.2040**	**0.5275**	**0.6360**	**0.5745**

study summaries of bug reports as shown in Figure 6, which will also reduce her time and effort as she no longer needs to read the whole bug reports.

VI. Experiment & Discussion

We divide our experiment into two sections: We measure the quality of time-sensitive keywords in terms of precision, recall, and F-measure, and we do a comparative analysis between a visualized and non-visualized bug report summary.

Evaluation of the Quality of Time-Sensitive Keywords: To assess the quality of the time-sensitive keywords, we compute mean precision, recall and F-measure of each topic in all of their time-segments, which are shown in Table IV. We need to check whether we can recover the original keywords of a topic by combining the keywords associated with each time segment. However, We consider a Term Frequency-Based (TF-Based) system as the baseline system for comparing the quality of time-sensitive keywords with our system. For each of the top five topics, we retrieve the top 10 keywords in each time segment both for our proposed prototype and for the TF-Based system. Then, we create a keyword union by combining all keywords retrieved from all of it's time segments, and separately compare each keyword union with the top 50 keywords derived by the LDA topic model for each topic. Table IV represents that in average our proposed prototype shows 53% precision, 64% recall and 57% F-measure and thus, outperforms TF-Based systems in all measures. It implies that our selected time-sensitive keywords for each topic segment are aligned with original topic-keywords far more than a TF-Based system.

Comparative Analysis of Visualized form of Bug Report Summary: Creating an extractive summary automatically in a consistent form is a difficult task. One approach is to determine a convenient way of providing contextual help to the developers, which we do. We conducted an ad-hoc mini user study with four graduate students of Computer Science Department at University of Saskatchewan, to see how well users accept the visualized bug report summary. In order to evaluate the effectiveness we designed a task of finding duplicated bug reports. Here, the participants were asked to identify duplicated bugs of a given new bug from a recommend list of existing candidate bugs. They were provided with either visualized or non-visualized bug report summary of each candidate bug and asked to study them during identifying duplicated bugs. The participants responded positively saying that such visualized summary is better than the non-visualized summary in gathering information from a bug report quickly.

Some comments are as follows: this seems like an innovative design, which has introduced to bug report management; it reduces my time and effort a lot. However, they also pointed some useful suggestions such as: the most important keywords could be highlighted both in summary and original bug report; a hyperlink capability into bug report summary sentences to the sentences in the original bug reports could be included.

VII. Conclusion and Future Works

We developed a number of visualizations to provide insightful information by employing topic analysis on a collection of bug reports. We provided information regarding topic evolution over time, searching by keywords, and visualizing bug report summaries. We found that topic evolution can aid either a developer or a project manager by showing important information. We also measured the precision, recall and F-measure of time-sensitive keywords and found that the approach outperforms a term frequency-based system.

The nature of our proposed prototype makes it very difficult to evaluate. Although the current evaluations and the user study show promise of the approach, we plan to conduct a fully-fledged user study with industry participants in order to fully explore the capabilities and limitations of the proposed approach. In this work, we used Eclipse bug reports for evaluating the approach. Although this should be enough to show the effectiveness of the prototype, we plan to evaluate it with bug reports from different domains. Furthermore, we plan to include more interesting and useful visualization features such as bug report duplication and localization. The user study with industry participants might suggest us more features to be included in the prototype.

References

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[2] K. Czarnecki, Z. Malik, and R. Lotufo. Modelling the Hurried Bug Report Reading Process to Summarize Bug Reports. In *Proc. ICSM*, pages 430–439, Sept 2012.

[3] T. Dal Sassc and M. Lanza. A Closer Look at Bugs. In *Proc. VISSOFT*, pages 1–4, Sept 2013.

[4] M. D'Ambros, M. Lanza, and M. Pinzger. "A Bug's Life" Visualizing a Bug Database. In *Proc. VISSOFT*, pages 113–120, June 2007.

[5] S. Havre, E. Hetzler, P. Whitney, and L. Nowell. Themeriver: Visualizing Thematic Changes in Large Document Collections. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):9–20, 2002.

[6] A Hora, N. Anquetil, S. Ducasse, M. Bhatti, C. Couto, M.T. Valente, and J. Martins. Bug Maps: A Tool for the Visual Exploration and Analysis of Bugs. In *Proc. CSMR*, pages 523–526, March 2012.

[7] L. Martie, V.K. Palepu, H. Sajnani, and C. Lopes. Trendy Bugs: Topic Trends in the Android Bug Reports. In *Proc. MSR*, pages 120–123, June 2012.

[8] F. Wei, S. Liu, Y. Song, S. Pan, X. M. Zhou, W. Qian, L. Shi, L. Tan, and Q. Zhang. TIARA: A Visual Exploratory Text Analytic System. In *Proc. KDD*, pages 153–162, July 2010.