What's hot and what's not: Windowed developer topic analysis

Abram Hindle
University of Waterloo
Waterloo, Ontario
Canada
ahindle@cs.uwaterloo.ca

Michael W. Godfrey
University of Waterloo
Waterloo, Ontario
Canada
migod@cs.uwaterloo.ca

Richard C. Holt
University of Waterloo
Waterloo, Ontario
Canada
holt@cs.uwaterloo.ca

Abstract

As development on a software project progresses, developers will naturally shift their focus between different topics and tasks many times. Managers and newcomer developers often seek ways of understanding what tasks have recently been worked on and how much effort has gone into each; for example, a manager might wonder what unexpected tasks occupied their team's attention during a period when they were supposed to have been implementing a set of new features. Tools such as Latent Dirichlet Allocation (LDA) and Latent Semantic Indexing (LSI) can be used to analyze check-in comments over the entire lifetime of a project. Previous work on developer topic analysis has leveraged these tools to associate commit log comments with independent topics extracted from these commit log comments. In this work, we use LDA and LSI to analyze periods, such as months, within a project's lifetime to create a time-windowed model of changing development topics. We propose a visualization of this model that allows us to explore the evolving stream of topics of development occurring over time. We demonstrate that windowed topic analysis offers advantages over topic analysis applied to a projects lifetime because many topics are quite local.

1 Introduction

What happened in the previous maintenance or development iteration? What were the developers working on? What was the developer in the next cubicle working on? Which common topics and issues were dealt with in the current release of the software? What were the topics of the previous release? Given the requirements agreed to at the start of the iteration did our developers work on them? What else did they work on?

Imagine that a development team had agreed upon a set of features to implement for a development iteration, but these features were not finished by the end of the iteration. The developers insist they focussed on those features, yet their manager is not sure what happened. In a scenario like this, topic analysis can help tease out the topics of development that the developers were focussed on. Topic analysis could highlight the orthogonal issues that developers were spending time on. Topic analysis extracts groups of words from documents (commit log comments) that are word distributions, topics, which characterize clusters of documents. These extracted topics often correlate with actual development topics that the developers discussed. By applying topic analysis locally to an iteration, both the developers and managers might be able to see what were the topics of development during that iteration.

Commits in SCSs can have multiple purposes. Thus we hope that our research will help uncover these multiple purposes that manifest themselves as topics interleaved within the commits. Others have demonstrated that topic analysis techniques, such as Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA), can aid in separating topics from these mixed comments. These topics often describe distinct ideas that are interleaved throughout the development. A topic represents a groups of commit log comments that are related to each other by their content. In this case a topic is a set of tokens.

Topic analysis can aid in partitioning a project's time-line into periods of developer focus. By breaking apart an iteration into sets of topics and trends (topics that recur), we may be able to recognize the underlying software development process from these commits. Alternatively, we can determine what particular maintenance task was being performed at a given time.

We propose to extend existing topic analysis techniques by applying them to windows of documents over time. We hope that by looking at topic clusters during periods or windows over time we can identify ongoing areas of focus on a select group of topics as well reflect more intermittent topics. Figure 1 illustrates the kind of visualization we want to automatically generate. We want to see topics that are unique to a certain time period, displayed across the time axis. Those topics that recur, or occur over a larger period are plotted continuously. In our example we have titled each topic with a single word.

In this paper we explore how topics shift over time in the source control system (SCS) of a project, using several open source database systems as examples. We analyze commit log comments in each time window and we extract the topics of that time window. We expect that topics will change over time and the similarities and differences in these topics will indicate developer focus and changes in developer focus as it changes over time.

Our contributions include:

- Demonstrating the value of windowed topic analysis using trends.
- Multiple visualizations of topics and trends over time.
- An exploratory case study using these techniques on several open source database systems.

2 Background

We now define some terms that we will use in the rest of the paper: A *message* is a block of text written by developers. In this paper, messages will be the CVS commit log comments made when the user commits changes to files in a repository. A *word distribution* is the summary of a message by word count. Each word distribution will be over the words in all messages. However, most words will not appear in a message, a word distribution is effectively a word count divided by the message size. A *topic* is a word distribution, a set of words that form a word distribution that is unique and independent within the set of documents in our total corpus. One could think of a topic as a distribution of the centroid of a group of messages. In this paper we often summarize topics by the top 10 most frequent words of their word distribution. A *trend* is a one or more similar topics that recur over time. Trends are particularly important, as they indicate long-lived and recurring topics that may provide key insights into the development of the project.

In terms of clustering and finding topic distributions, Latent Dirichlet Allocation (LDA) [1] competes with Latent Semantic Indexing (LSI) [6, 7], probabilistic Latent Semantic Indexing (pLSI) [1] and semantic clustering [3, 4]. These tools are used for document modelling, document clustering and collaborative filtering. LDA attempts to encode documents as a mixture model, a combination of topics. LDA is used in software engineering literature [9, 5]to extract topics from documents such as methods, bug reports, source code and files.

LDA, LSI and semantic clustering extract topic clusters from the documents that are independent from one another. LDA does not name these clusters but the clusters consist of words whose relationship is often obvious to someone familiar with the corpus. We could swap LDA for LSI or semantic clustering and likely produce similar results. Our data is posed as documents with word distributions (word counts per documents) and LDA extracts distinct topics (clusters of words) from the documents.

2.1 LDA, LSI and Semantic Clustering

In order to infer or associate the expertise of an author with topics extracted from SCS, Linstead et al. proposed an author-source-code model using LDA [5]This model is essentially an extension of the author-topic (AT) model [8], which associated authors and topics extracted with LDA.

Lukins, Kraft and Etzkorn [9] use LDA to help bug localization by querying for documents that are related to a bug's topic. They used LDA to build a topic model and then queried against it using sample documents. These queries would indicate the proportion of topics that were associated with the query and the related documents.

LSI is related to LDA and has been used to identify topics in software artifacts for formal concept

analysis and concept location [6, 7] Concept analysis aims to automatically extract concepts from source code or documents that are related to the source code. Concept location concerns how high-level concepts relate to low level entities such as source code. For example, when fixing a bug, how and where do the concepts in the bug report map back to the source code? Semantic Clustering has also been used for similar purposes [3, 4] as it is similar to LSI.

Grant et al. [2] have used an alternative technique, called Independent Component Analysis to separate topic signals from software code.

Our technique differs from the previous techniques because we apply LDA locally to month-long windows of commit log comments, whereas other approaches apply LDA once to the entire project. This allows us to examine the time-based windowing of topics over its development history.

3 Preliminary Case Study

In our first exploratory pass we wanted to see if LDA could provide interesting topics extracted from a real system. We took the repository of MySQL 3.23, extracted the commits, grouped them by 30 day non-overlapping windows, and then applied LDA to each of these windows. We asked LDA to make 20 topics per window and we then examined the top 10 most frequent words in that topic. We found there were common words across topic clusters, such as *diffs*, *annotate* and *history*; these words probably should be viewed as stop words. There were notable transitional topics such as in the first window the word *BitKeeper* appears because MySQL adopted Bitkeeper for their source control system yet in the following windows there were no mentions of BitKeeper. *RENAME* also only appeared once in the first window and never again. Per each topic we tried to identify the purpose of the topic; to our surprise we found that even with only a little familiarity with the code base that naming the topic was straightforward.

A sampling of the notable topic words is displayed in Table 1, we chose topics that we felt confident we could name. To name each topic we selected a term that seemed to best summarized that topic. After extracting these topics, we attempted to track the evolution of topics by visualizing the topics and joining similar topics into trends. Figure 1 displays a manually created sample plot of the extracted topics in Table 1.

2000	Jul	chmod
2000	Sep	fixes benchmark logging Win32
2000	Nov	fixes insert_multi_value
2001	Jan	fixes Innobase Cleanups auto-union
2001	Mar	bugfix logging TEMPORARY
2001	Jul	TABLES update Allow LOCK
2001	Aug	TABLES row version
2001	Sep	update checksum merge

Table 1. Sampled topics from MySQL 3.23, some with continuous topics. These tokens were pulled from LDA topic. Each token is a summary of one LDA generated topic from MySQL 3.23 commit comments.

4 Methodology

Our methodology is to first extract the commits from SCS repositories such as CVS or Bitkeeper. Then we extract the revisions and commits, grouping the revisions into commits if necessary. We extract the commit log comment of each commit, count the occurrence of each word in the message, remove stop words, and then produce word distributions for each message.

4.1 Extraction of Repositories and Documents

We mirrored the repositories and their revisions using software such as rsync, CVSSuck, `softChange` and bt2csv. `softChange` provided a general schema for storing revisions. CVSSuck and rsync mirrored CVS repositories while bt2csv mirrored web accessible BitKeeper repositories.

The documents we are analyzing are the commit log comments i.e., the comments that are added when revisions to the project are committed. Documents are converted into word count distributions. These distributions are then normalized by the size of the message. After all messages are processed the distributions are extended to include all words from all of the distributions.

4.2 Windowed Sets

Given all messages, we group the messages into windows. We could use overlapping windows, but in this paper we use non-overlapping windows of a set time unit because it simplifies analysis. Windowing by time allows for many levels of granularity. We usually used one month as the length of our

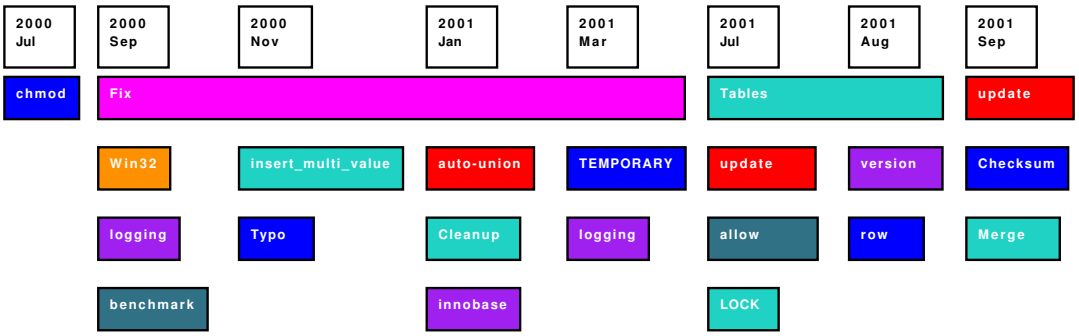

Figure 1. Example of topics extracted from MySQL 3.23. This is the kind of plot we eventually want to illustrate: named topics and topic trends. The horizontal axis is time by month examined. The vertical axis is used to stack topics that occur at the same time. Longer topics are topics which recur in adjacent windows. Colors are arbitrary.

time windows. While we could use different window lengths for the study we think that a month is a sizable unit of development, which is large enough to show shifts in focus, but coarse enough not to show too much detail.

4.3 Apply Topic Analysis to each Window

Once we have our data windowed, we apply our Topic Analysis tool to each window and extract the top N topics. We used 20 topics in our case studies. Our Topic Analysis tool is an implementation of LDA, but we could have used other similar tools like LSI, but previous work showed more promising topic analysis results with LDA. We note that this step is a slow one, as executing even one instance of LDA involves significant computation, and we perform LDA once per window.

4.4 Topic Similarity

Once we have our topics extracted for each window, we analyze them and look for topics that recur across windows. Our input for topic similarity is the top 10 most common words of each topic.

We then cluster these topics into trends by comparing them to each other. Each topic is compared to each other topic in the system, given a certain threshold of top 10 similarity, such as 8 out of 10 words. Then given this topic similarity matrix, we find the transitive closure of similar topics; that is, if we model topics are nodes and arcs as similarity, we fill flood along the similarity arcs until we have partitioned the topics into clusters of similar topics. Figure 3 illustrates clustering of topics by topic similarity. These clusters of topics are called trends. A trend is probably interesting if it contains more than one topic.

This technique has weaknesses in that nodes that are a few neighbors away in similarity might not share any similar words. We use this measure because we want to be able to color or highlight topics that are related to each other and track them throughout time.

Once we have determined our similarity clusters we can choose to analyze and to plot the topics.

4.5 Visualization

We have devised several techniques for visualizing these topics and trends. For all of these techniques if we find trends that have continuous segments, then we plot those segments as joined horizontally across time. One technique is the *compact-trend-view*, shown in Figure 4 and Figure 5, that displays trends as they occur on the time-based x-axis (placement along the y-axis is arbitrary). Another technique is the *trend-time-line*, shown in Figure 6, each trend gets it own distinct y-value, while the x-axis is time; these topics are then each plotted on their own line across time as they occur. Our final technique is the *trend-histogram*, shown in Figure 7 where we plot each trend on its own line but stack up the segments of the trend, much like a histogram.

The *compact-trend-view* (Figure 4) tries to sink

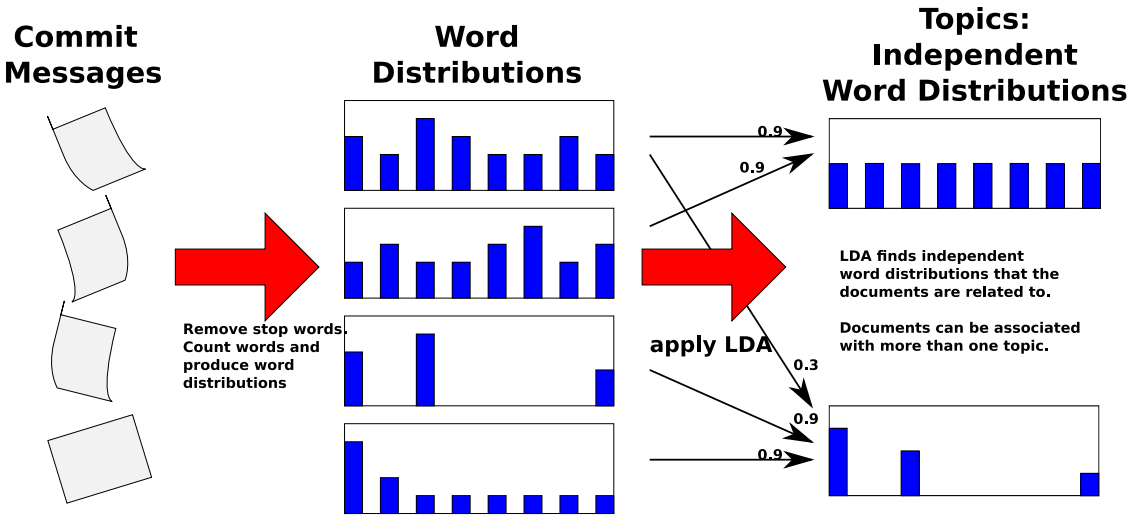

Figure 2. How commits are analyzed and aggregated into Topics and Trends. Commits are first extracted, then abstracted into word counts or word distributions. These word distributions are then given a topic analysis tool like LDA. LDA finds independent word distributions (topics) that these documents are related to.

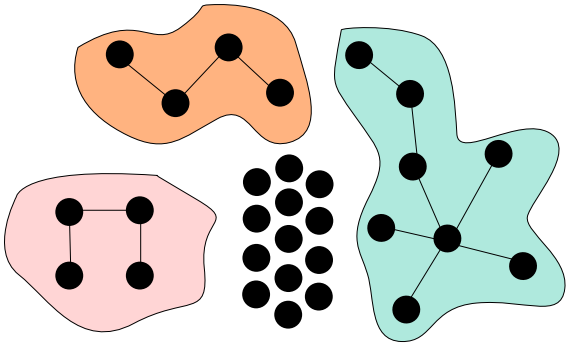

Figure 3. Topic similarity demonstrated by clustering topics by the transitive closure (connectedness) of topic similarity. Nodes are topics and arcs imply similarity. Similarity could be a measure such as topics share 8 out of 10 top ten words.

the larger trends to the bottom. Once the larger trends are stacked we fill in the gaps with the smaller trends in the same window, and then stack the smaller trends on top. Although there is a chance that gaps will be left due to non-optimal stacking in practice there are lot of small trends (90% of all trends contain 1 topic) that fill in these gaps quickly. Different instances of the same trend share the same color; apart from that, the color of trends is randomly chosen and color similarity is not meaningful. The compact-trend-view is convenient because it shows all of the topic information, as shown by the zoomed-in view of MaxDB 7.500's compact-trend-view in Figure 5. This view makes repeating continuous trends easy to pick out, although discontinuous trends are harder to spot.

The *trend-time-line* (Figure 6) displays repeating trends more clearly by dedicating a horizontal line for trend segments belonging to one trend. This means that if a trend contains discontinuous segments then the segments appear on the same line. However, the least common trends need to be pruned away or the view will be very long.

The *trend-histogram* (Figure 7) superficially resembles the trend-time-line. However, in this view the trends are plotted together by stacking to the left of their row, thus time information is lost. The trends are ordered by the number of topics in the trend. The trend-histogram shows the count of instances of a trend and thus indicates which trends occur the most. Unfortunately due to the large number of topics, given the allotted space, it is often best to crop off the trends with only one topic, otherwise the tail is long.

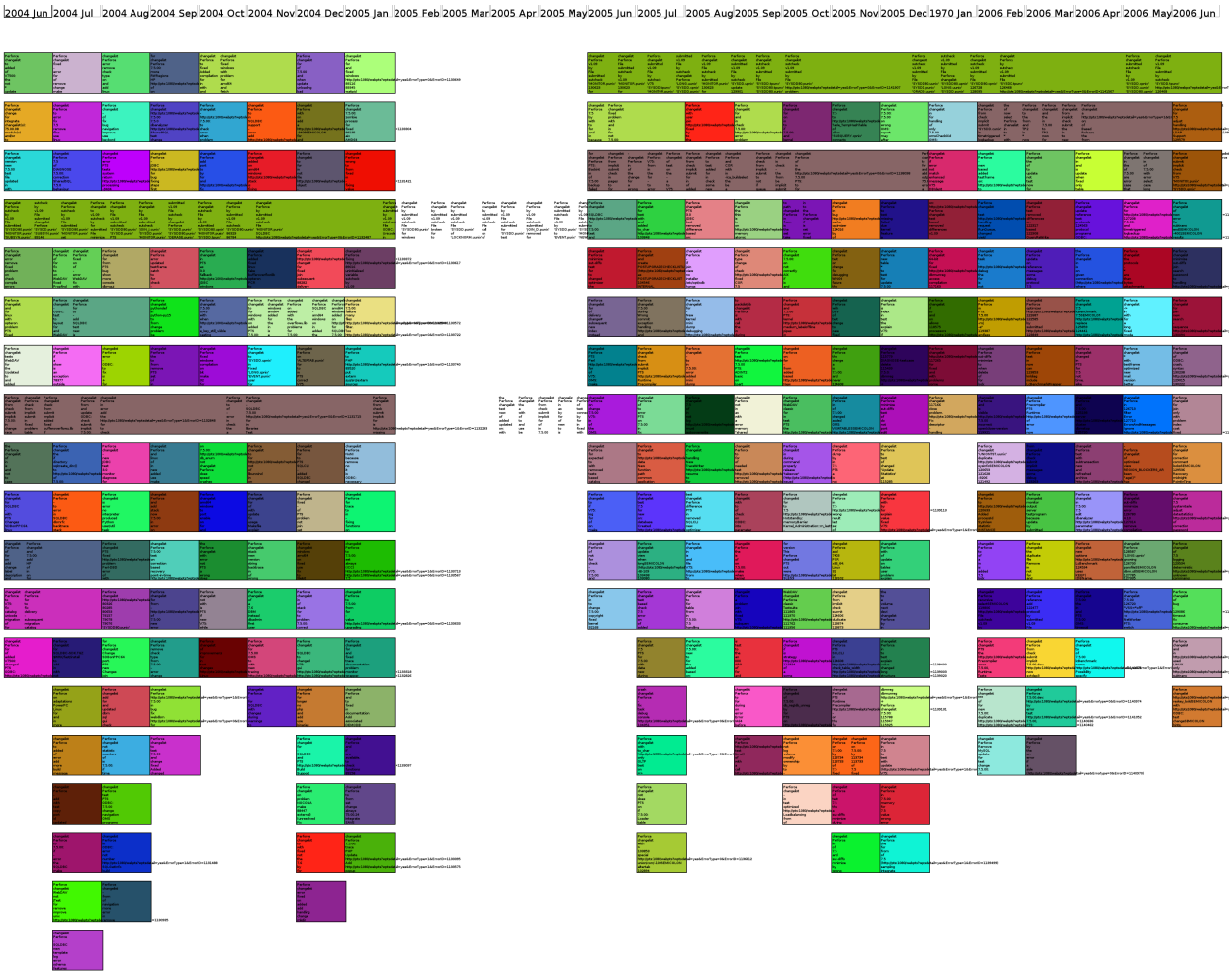

Figure 4. Compact-trend-view shows topics per month for MaxDB 7.500. The x-axis is time in months, the Y axis is used to stack topics occurring at the same time. Trends that are continuous are plotted as continuous blocks. The top 10 words in the topics are joined and embedded in box representing the topics.

Figure 5. A zoomed in slice of compact-trend-view of topics per month of MaxDB 7.500. The topic text is visible in each topic box. Trends are plotted across time continuously.

Figure 6. Trend-time-line: Trends plotted per month of MaxDB 7.500. Time in months are plotted along the x-axis, each row on the y-axis is associated with a trend ranked by size in descending order

5 Results

We applied our methodology to multiple database systems that we extracted. To analyze these extracted repositories we used: Hiraldo-Grok, an OCaml-based variant of the Grok query language; Gnuplot, a graph plotting package; lda-c, a LDA package implemented by Blei et al. [1]; and our trend plotter, implemented in Haskell.

We applied our tools and visualizations to the repositories of several open source database systems: PostgreSQL, MaxDB and Firebird.

5.1 PostgreSQL

When we examined PostgreSQL, we did not find many trends with two or more topics, using a similarity of 7/10. Those trends that we did find were not very large, lasting only 3 months at most. The first and second largest trends were dedicated to the work of two external developers: Dal Zotto, Dan McGuirk. The fifth largest trend related to work by PostgreSQL developer D'Arcy. Other topics of the larger trends were changes to the TODO, and time string formating topics relating to timezones.

If we kept the stop words we found that the large trend consisted mostly of stop words and non-stop words such as *patch*, *fix*, *update*. By decreasing the similarity constraint to 1/2 similarity the largest most common trend, which stretched across the entire development, contained these same words (*patch*, *fix*, *update*). The second largest trend mentions Dal Zotto, while the third largest trend men-

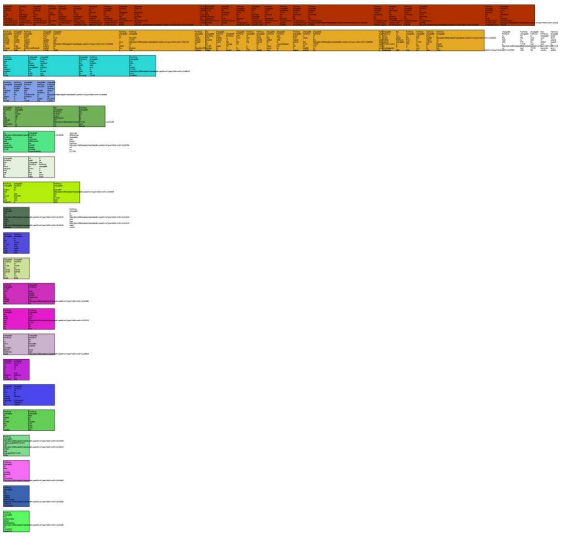

Figure 7. The top part of a trend-histogram of MaxDB 7.500, ordered by topic occurrence. X-axis determines the number of continuous months of a trend. Trends are ranked by the number of topics that a trend contains in descending order.

tions the [PATCHES] mailinglist and the names of some patch contributors. Other repeating topics refer to portability with Win32, Central Europe Time (CEST) from email headers, issues with ALTER TABLE and CVS branch merging, because CVS does not record merges explicitly.

5.2 Firebird

We tracked Firebird from August 2000 to January 2006. We found that with a similarity of 7/10 that Firebird had far more continuous and recurring trends than PostgreSQL. The first large trend was segmented across time but explicitly references one author carlosga05 and words like *added*, *fixed* and *updated*.

The second largest trend was during the month of March 2001. It was related to incremental building and the porting of developer Konstantin's Solaris port of the Firebird build files. The third largest trend was about JDBC, which is how Firebird and Java communicate. Other trends included topics regarding AIX PPC compilation, updating the build process, internationalization and UTF8, Darwin build support and bug fixing.

Topics that were not trends but appeared to be interesting were mostly external bug fixes submitted to the project. In these cases, the developers would express gratitude in their commit log comments, such as "Thanks, Bill Lam". Other easily discernible topics included tokens and phrases such as: *compiler workarounds*, *nightly updates*, *packets* and *MSVC++*.

5.3 MaxDB 7.500

The plots we produced of MaxDB 7.500 were unlike those of the other systems, as there was a period where no real development occurred and thus there were no topics or trends whatsoever (see the gap in Figure 4). Using a topic similarity of 7/10 we evaluated MaxDB 7.500. MaxDB 7.500's first period was from June 2004 to January 2005, and its second period was from June 2005 to June 2006.

The largest common trend has references to build system files like SYSDD.punix and MONITOR.punix. This trend is partially displayed at the top of the zoomed in compact-trend-view (Figure 5) and at the top of the trend-histogram (Figure 7). Other tokens mentioned are *Sutcheck v1.09* (the prefix SUT stands for Storable Unit Type), *Sutcheck* is a tool that would also automate check-ins using a Perforce SCS tool, which was exporting check-ins to CVS.

The second largest common trend seems to be a side effect of an automated check-in that is annotated as "implicit check-in" (see the bottom of Figure 5). These were check-ins that were produced when importing changes from an external Perforce repository.

The third most common trend, seen on Figure 6, seemed to include tokens related to operating system support, such as *Linux* and *Windows*, as well as architecture support, *AMD64* and *Opteron*. The word *problem* was common among all of these trends. This trend seemed related to the smaller fourth largest trend that had tokens *AMD64* and *Windows*. This example shows that topics can overlap but still not match via our similarity measure.

Bug tracker URLs dominated unique topics during some months. For instance in the last month of MaxDB 7.500 development every topic contained 1 unique Bug tracker URL, this pattern did not occur in the previous month. We investigated the revisions and we did find there was a difference in behavior between, and that the last month of development commits were mostly referencing the bug tracker. If the topics of one month were about unique bug tickets being addressed, the global topic analysis would probably miss this, yet these bug tickets were the focus of development for that month but not necessarily globally relevant.

The query optimizer was a topic that recurred during MaxDB's development. In our plots, topics that mention *optimizer* occur four times, yet in the global-trend-view (Figure 8) it is not in any of the topics. A query optimizer is rather important to an DBMS, but as we have shown it just does not pop up as a topic on its own. We tried to remove words to see if we could get an *optimizer* topic. After removing stop words and then two of the most common words, the global analysis finally had a topic with optimizer in its topic ten words. This shows that optimizer was important but it had been obscured, while it would have been noticed using the more locally relevant approach that we propose.

We noticed that commits that mentioned *PHP* occurred two thirds less frequently than commits that mentioned *Perl*, but *Perl*-related topics appeared in the global static topics for MaxDB while *PHP*-related topics did not. Our local topic analysis mentioned *PHP* in 5 different topics, yet only mentioned *Perl* in 4 different windowed topics and 1 global topic. Perhaps this is because there was a cluster of *Perl* mentions during one month while

the *PHP* mentions were more spread out.

We noticed that *Perforce* and *Changelist* were in just about every topic, so we added them to the stop words list. As a result, longer trends were shortened and sometimes the total number of topics found per month were less. Evaluating different similarity thresholds showed that by removing common words one reduces the general similarity of topics. That said, the larger topics were still clearly apparent. Thus if more relevant stop words are added one should tune the topic similarity parameters to handle these changes.

5.4 Compare with topics over the entire development

Previous work on topic analysis that employed LSI and LDA typically extracted a specified number of topics and then tracked them over time. This means that a single, static set of topics was tracked across the whole development history of a project. *Global topic analysis* is topic analysis applied against the entire lifetime of a project.

We carried out the same kind of analysis, to compare the results against our time-windowed approach. To produce figure 8, we extracted 20 topics and plotted the number of messages per month that were related to that topic. We found was that one topic would often dominate the results, as shown in the third row of Figure 8, while messages related to other topics did not appear often. This approach seems reasonable if most of the extracted topics are of broad interest during most of the development process. However, it may be that some topics are of strong interest but only briefly; in such a case, a window-based model gives a much stronger indication of the fleeting importance of such topics, and can help to put such a topic into its proper context.

If we approach the difference of global topic analysis and windowed topic analysis via common tokens we can see that common tokens tend to dominate both results. For MaxDB 7.500, our local topic approach netted us topics that contained these relatively important and common words, which did not occur in the topics produced by global topic analysis: *UNICODE*, *DEC/OSF1*, *ODBC*, *crash*, *SQLCLI*, *SYSDD.cpnix*, *backup*, *select*, *make*, *memory*, *view*, and finally *debug*. *ODBC* is an important topic token, because it often determines how databases communicate with software. None of these tokens were part of the global topic analysis topics, but they were part of 566 commits (6% of the entire system) to MaxDB 7.500. These tokens were part of 87 out of 520 (26 months, 20 topics per month) of our locally generated topics.

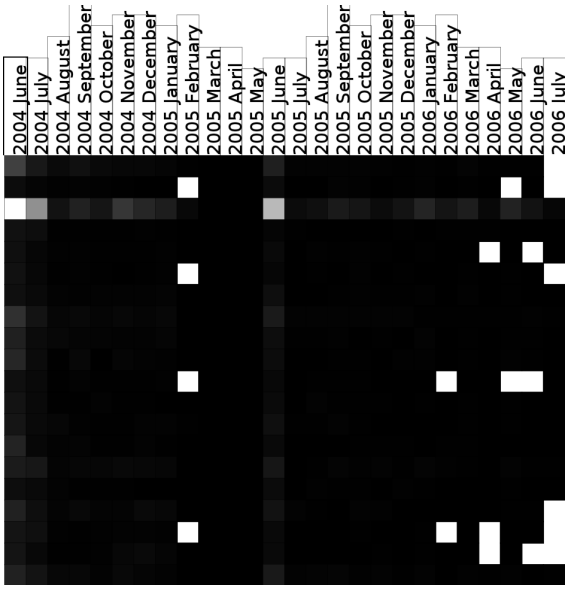

Figure 8. MaxDB 7.500 topics analyzed with global topic analysis, 20 topics (each row) and their document counts plotted over the entire development history of MaxDB 7.500 (26 months). The shade of the topic indicates the number of documents matching that topic in that month relative to the number of documents (white is most, black is least).

Even with our liberal topic similarity metrics that produced both long and short trends, we showed that there are only a few trends in a repository that recur. Since so few trends recur and so many trends appear only once this suggests that global topic analysis might be ignoring locally unique topics.

The utility of global topic analysis is questionable if the value of information decreases as it becomes older. Perhaps older trends will dominate the topic analysis. Windowed localized topic analysis shows what are the unique topics, yet seems to give a more nuanced view of the timeliness of the important topics.

6 Validity Threats

In this study we are explicitly trusting that the programmers annotate their changes with relevant information. We rely on the descriptions they provide. If the language of check-in comments was automated we would be analyzing only that.

We truncated the topics to top 10 tokens, so this summary of a topic might not have been as useful as determining the actual topic distance between two word topic distributions. By truncating tokens we could be losing valuable data.

Each project seemed to have their own set of stop words, and we did not remove them, but perhaps dropping some of these stop words would aid the clarity of topics produced through topic analysis; alternatively it might cause our topic similarity plots to be fundamentally different.

The number of commits per month is inconsistent as some months have many changes while other months have almost none.

7 Conclusions

We proposed and demonstrated the application of windowed topic analysis, that is, topic analysis applied to commit messages during periods of development. This approach differs from previous work that applied topic analysis globally to the entire history of a project without regard to time. We showed that many topics that exist locally are relevant and interesting yet would often not be detected via global topic analysis. We identify recurring topics with a topic similarity measure that looks for topics which recur and mutate repeatedly throughout the development of the software project.

The value of windowed topic analysis was demonstrated on a case study of three database systems with a particular focus on MaxDB 7.500. We showed that global topic analysis missed important topics such as *ODBC*.

We presented several visualization techniques that focused on different aspects of the data: temporality of trends, trend size, and a compact-trend-view. The compact-trend-view shows much more information than the views that global analysis could show, it indicates how focused a period is by the total number of topics, as well, it shows topics by similarity so one can track trends across time. Our trend-histogram immediately highlights and measures the size of trends while our trend-time-line view shows how a topic reoccurs over time. These visualizations help us get an general feeling about the common and unique topics that developers focus on during development. If implemented properly a user could easily zoom in and query a summary of a topic or trend.

7.1 Future Work

One avenue of future work is automatic topic labelling. Given a word distribution we should be able to automatically select a word or term that describes that distribution. Potential external sources of topic names include: software engineering taxonomies, ontologies and standards.

References

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

[2] S. Grant, J. R. Cordy, and D. Skillicorn. Automated concept location using independent component analysis. In *15th Working Conference on Reverse Engineering*, 2008.

[3] A. Ko, B. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 127–134, Sept. 2006.

[4] A. Kuhn, S. Ducasse, and T. Girba. Enriching reverse engineering with semantic clustering. *Reverse Engineering, 12th Working Conference on*, pages 10 pp.–, Nov. 2005.

[5] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining eclipse developer contributions via author-topic models. *Mining Software Repositories, International Workshop on*, 0:30, 2007.

[6] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic. An information retrieval approach to concept location in source code. *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, pages 214–223, Nov. 2004.

[7] D. Poshyvanyk and A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. *International Conference on Program Comprehension*, 0:37–48, 2007.

[8] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth. The author-topic model for authors and documents. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 487–494, Arlington, Virginia, United States, 2004. AUAI Press.

[9] L. H. E. Stacy K. Lukins, Nicholas A. Kraft. Source code retrieval for bug localization using latent dirichlet allocation. In *15th Working Conference on Reverse Engineering*, 2008.