What's Hot and What's Not: Windowed Developer Topic Analysis

Abram Hindle, Michael W. Godfrey and Richard C. Holt
University of Waterloo
Waterloo, Ontario
Canada
{ahindle,migod,holt}@uwaterloo.ca

Abstract

As development on a software project progresses, developers 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 new features. Tools such as Latent Dirichlet Allocation (LDA) and Latent Semantic Indexing (LSI) can be used to extract a set of independent topics from a corpus of commit-log comments. Previous work in the area has created a single set of topics by analyzing comments from the entire lifetime of the project. In this paper, we propose windowing the topic analysis to give a more nuanced view of the system's evolution. By using a defined time-window of, for example, one month, we can track which topics come and go over time, and which ones recur. We propose visualizations 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 project's lifetime because many topics are quite local.

1. Introduction

Software managers know that — during any given development cycle — there is usually a difference between the set of tasks that developers are supposed to be working on and the tasks they are actually working on. To better understand the realities of development, it would be helpful to be able to answer questions such as: What topics dominated the previous release cycle? Given the requirements agreed to at the start of the iteration how much work did our developers spend on them? What other tasks did they work on? Which topics seem to recur over and over between releases? How long do different topics persist? Are there latent cause-effect relationships between different topics?

Topic analysis uses tools such as Latent Dirichlet Allocation (LDA) and Latent Semantic Indexing (LSI) to extract independent word distributions (topics) from documents (commit log comments) [11, 12, 13, 10]. Ideally these extracted topics correlate with actual development topics that the developers discussed during the development of the software project. Topic analysis often allows for a single document, such as a commit message, to be related to multiple topics. Documents represented as a mixture of topics maps well to commits to source code, which often have more than one purpose. A topic represents both a word distribution and a group of commit log comments that are related to each other by their content. In this case a topic is a set of tokens extracted from commit messages found within a projects source control system (SCS).

Previous work on topic analysis has been applied to the entire history of a project [9, 13, 12] to produce a single set of topics. In this paper we explore the idea of time-windowed topic analysis; that is, we pick a period of, say, one month and perform topic analysis within each window. In addition to being able to study just which topics were actually worked on and when, we can also analyze the historical timeline of topics for patterns and trends. For example, we might wish to categorize each topic as being local to a single time window, spanning multiple consecutive windows, or recurring periodically. Alternatively, we might explore evolutionary patterns, such as if topics often recur together, if one particular topic usually precedes another, if unplanned topics outnumber planned topics, if particular topics recur near the end of development iterations, etc.

As part of this work, we wish to explore approaches to automatically generating visualizations of the topic trends and development timelines, so that managers may be able to more easily grasp the topic-related activities of the developers. Figure 1 gives an example. Those topics that recur, or occur over a larger period are plotted continuously. In our example we have titled each topic with a word chosen from its word distribution. Given a visualization, such as Figure 1, 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 and maintenance tasks from these commits.

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. We will show that most topics are locally relevant, i.e., relevant to a single window of time. Our windowing approach supports both local (single window) analysis and global (entire history) analysis.

Our contributions include:

- We introduce windowed topic analysis, and demonstrate its utility compared to global topic analysis.
- We present a number of visualizations of topics and their trends over time to aid communication and analysis of these topics and trends.
- We provide 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 a developer. In this paper, messages will be the CVS and BitKeeper 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 each message. A word distribution is effectively a word count divided by the message size. A *topic* is a word distribution, i.e., 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 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) [2] competes with Latent Semantic Indexing (LSI) [11, 12], probabilistic Latent Semantic Indexing (pLSI) [2] and semantic clustering [7, 8]. 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 has been used in software engineering literature [13, 10] 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. The clusters are not named by the process, and consist of words whose relationship is often obvious to someone familiar with the corpus. For our purposes, 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 [10]. This model associates authors and topics via their associated documents.

Lukins et al. [13] 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 searched for similar documents by querying with sample documents.

LSI is related to LDA and has been used to identify topics in software artifacts for formal concept analysis and concept location [11, 12]. 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, such as bugs, relate to low level entities such as source code. Semantic Clustering has also been used for similar purposes [7, 8] as it is similar to LSI.

Grant et al. [5] and have used an alternative technique, called Independent Component Analysis [3] to separate topic signals from software code.

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

Imagine that we are analyzing one month of development in a project where there were three major tasks: bug fixing, adding a new GUI, and documentation updates relating to the system and the GUI. To use LDA we would get the word distributions of each commit message, and have the LDA tool extract a set of topics from the data. Ideally, the LDA analysis would return four topics, corresponding to bug fixing, GUI implementation, documentation, and "other". Each topic essentially corresponds to a word distribution. For example, the bug fixing topic might include words such as *bug*, *fix*, *failure*, and *ticket*, while the GUI topic might include *widget*, *panel*, *window*, and *menu*, and the documentation topic might include *section*, *figure*, and *chapter*. The last topic would include words that were not commonly found in bug reports, GUI fixing, or documentation commits; that is, the LDA analysis would put into this last category the words that were independent of the other topics it found.

LDA is an unsupervised technique. The user provides a corpus of documents and specifies various parameters, and then LDA automatically generates a set of likely topics based on word frequencies. It is then up to the user to decide if the topics are meaningful by looking at the most frequent words in each and deciding what each discovered topic should be named.

In addition to finding the topics, LDA can also associate the documents (i.e., commit messages) in its corpus to one or more of the topics it discovers. For example, a commit of a simple bug fix would probably contain a word such as *ticket*, and thus heavily associate with the bug fix topic. A commit that concerned user documentation of the GUI would likely be associated with both the documentation and GUI topics if it shared words from both topics. And a commit that did not concern bugs, GUIs, or documentation would likely be related to the "other" topic.

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 chose one month because it was smaller than the time between minor releases but large enough for there to be many commits to analyze. We chose 20 topics because past experimentation showed that fewer topics might aggregate multiple unique topics while any more topics seemed to dilute the results and create indistinct topics. We chose the top 10 words because we wanted to be able to distinguish topics even if some common words dominated the topics.

We found there were common words across topic clusters, such as *diffs*, *annotate* and *history*. Words like these could be treated as stop words, since they occur across many of the topics and since they relate to the mechanisms of version control rather than the semantic content of the commit. There were topics which appeared only during some transitional period and never again, such as *RENAME* and *BitKeeper*. *BitKeeper* which appears when MySQL switched

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 the top 10 most common words found in LDA extracted topics. Each token is a summary of one LDA generated topic from MySQL 3.23 commit comments.

to BitKeeper for their SCS. For each topic we tried to find a word to describe its purpose. To our surprise we found that even with only a little familiarity with the code base that naming the topic was straightforward. To find the purpose of a commit we looked at the most frequent words in the word distribution of the topic and tried to summarize the topic, then we looked into the commits related to that topic to investigate if we were correct; since the commit messages and the word distribution share the same words the purpose extracted from the top 10 words was usually accurate.

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 summarize 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 plot of the extracted topics in Table 1.

4. Methodology

Our methodology is to first extract the commit log comments from a project's SCS repository. We filter out stop words and produce word distributions from these messages. These distributions are bucketed into windows, and then each window is subject to topic analysis and then we analyze and visualize the results. Figure 2 depicts the general process for processing and analyzing the commit messages.

4.1. Extraction of Repositories and Documents

We mirrored the repositories and their revisions using software such as rsync [14], CVSsuck [1], `softChange` [4], and bt2csv [6]. `softChange` provided a general schema for storing revisions and grouped

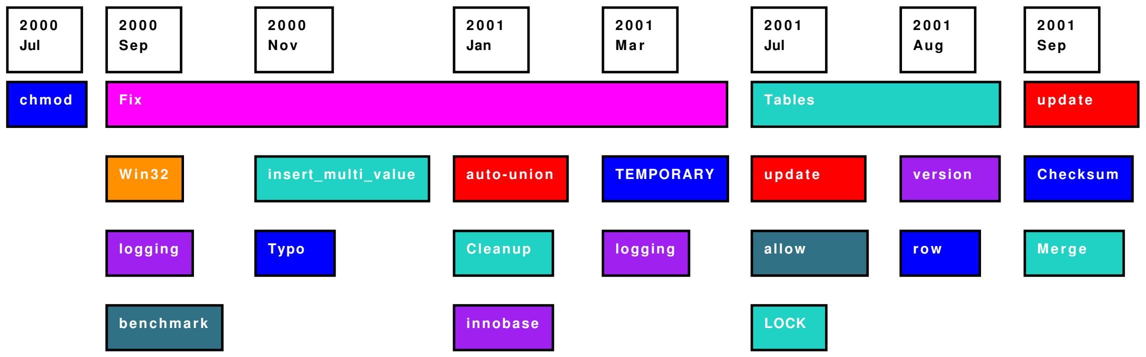

Figure 1. Example of topics extracted from MySQL 3.23. The horizontal axis is time by month examined. The vertical axis is used to stack topics that occur at the same time. Longer topics are topics which recur in adjacent windows. Colors are arbitrary.

CVS revisions into commits. 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. For each commit log comment we produce word distributions by counting the occurrence of each word in the message, removing stop words and then normalizing the distribution by the size of the message, in tokens. 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. Overlapping windows would increase the likelihood of trends, but for this study we lacked the space and were more concerned if topics ever repeated, overlapping might skew that result. Windowing by time allows for many levels of granularity. We used one month as the length of our time windows. While we could use different window lengths for this 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. Choosing a month as the window provided us with enough documents to analyze.

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 decided to use a threshold of $N = 20$ topics based on our previous experience with LDA; we found that using more than 20 makes the boundaries between topics less distinct.

Our Topic Analysis tool is based around a third-party implementation of LDA. While we could have used LSI or similar techniques as the engine, our preliminary studies found LDA to provide more promising results for topic analysis. 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. Figure 2 shows how LDA is applied to a set of messages, and how the topics are extracted and related to the messages.

4.4. Topic Similarity

Once we have our topics extracted for each window, we analyze them and look for topics that recur across windows. We then cluster these topics into trends by comparing them to each other using topic similarity.

Our input for topic similarity is the top 10 most common words of each topic. Each topic is compared to each other topic in the system, given a certain threshold of similarity, such as 8 out of top 10 matching words. 10 words were chosen because often people care about top 10 comparisons and it allowed for some common words to exist yet not produce too many trends. We then apply the transitive closure on similar topics to our set of topic similarities; this is similar to modelling topics as nodes and similarity as arcs, then fill flooding 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*. Trends indicate that a topic has occurred over more than one period during development.

This approach does have a weakness, in that nodes that are a few neighbors away in similarity might not share any

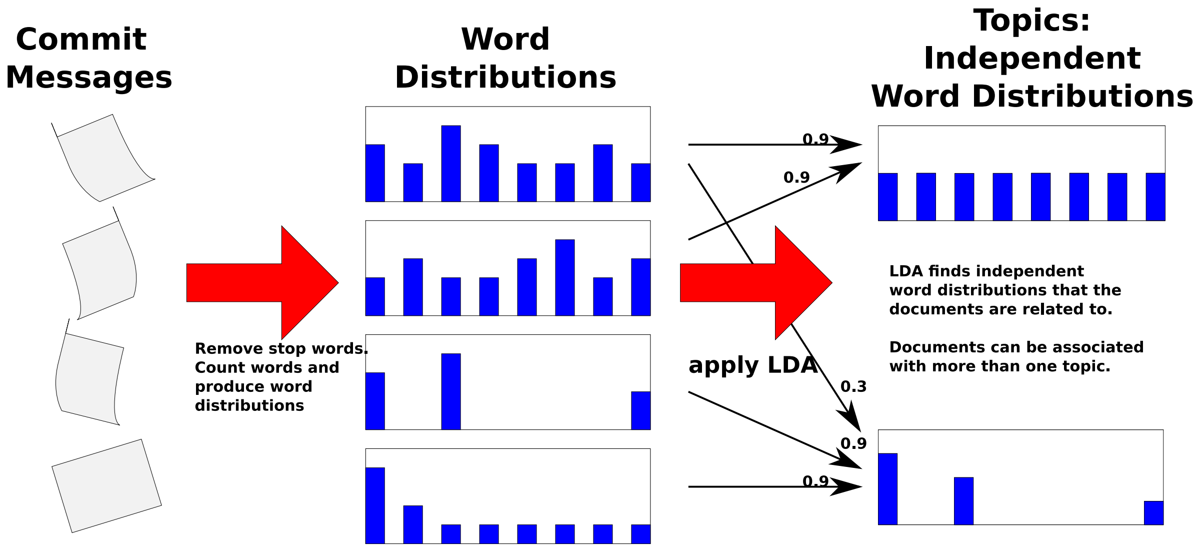
Commit Messages
Word Distributions
Topics: Independent Word Distributions
Remove stop words. Count words and produce word distributions
apply LDA
0.9
0.9
0.3
0.9
0.9
LDA finds independent word distributions that the documents are related to.
Documents can be associated with more than one topic.

Figure 2. How commits are analyzed and aggregated into topics and trends: commits are first extracted, then abstracted into word counts or word distributions which are then given to a topic analysis tool like LDA. LDA finds independent word distributions (topics) that these documents are related to (the numbers indicate similarity between documents and topics).

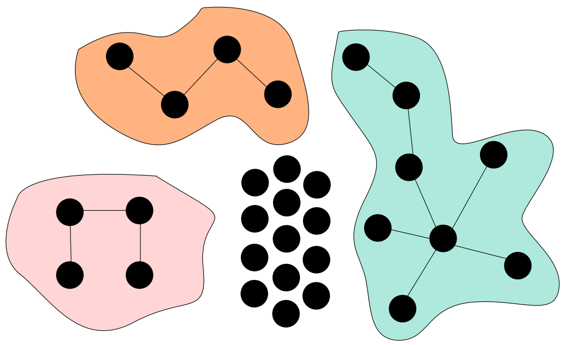

Figure 3. Topic similarity demonstrated by clustering topics by the transitive closure (connectedness) of topic similarity. Nodes are topics and arcs imply some notion of similarity, e.g., topics share 8 out of top 10 words.

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 are ready to analyze and to plot the topics.

4.5. Visualization

Visualization is an integral part of our topic analysis which allows us to quickly explore the topics and trends of a project. Visualizing the results allows us to explore the data from different points of view to address different questions: How are the topics spread over time? How many independent topics per period were there? What were the repeating trends? Did the trends dominate, or did local topics dominate? How contiguous were the trends? To address these questions we will describe and employ multiple visualizations.

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, where each trend gets its 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. Each topic has its top 10 words listed in descending order of frequency from top to bottom, this text is embedded inside the topic's box, which at this resolution requires the reader to zoom in electronically. A trend has all of its topic text embedded side by side within the same trend box.

The *compact-trend-view* (Figure 4) attempts to show all topics and trends at once across time in a compact view that could fit on one page. The *compact-trend-view* (Figure 4)

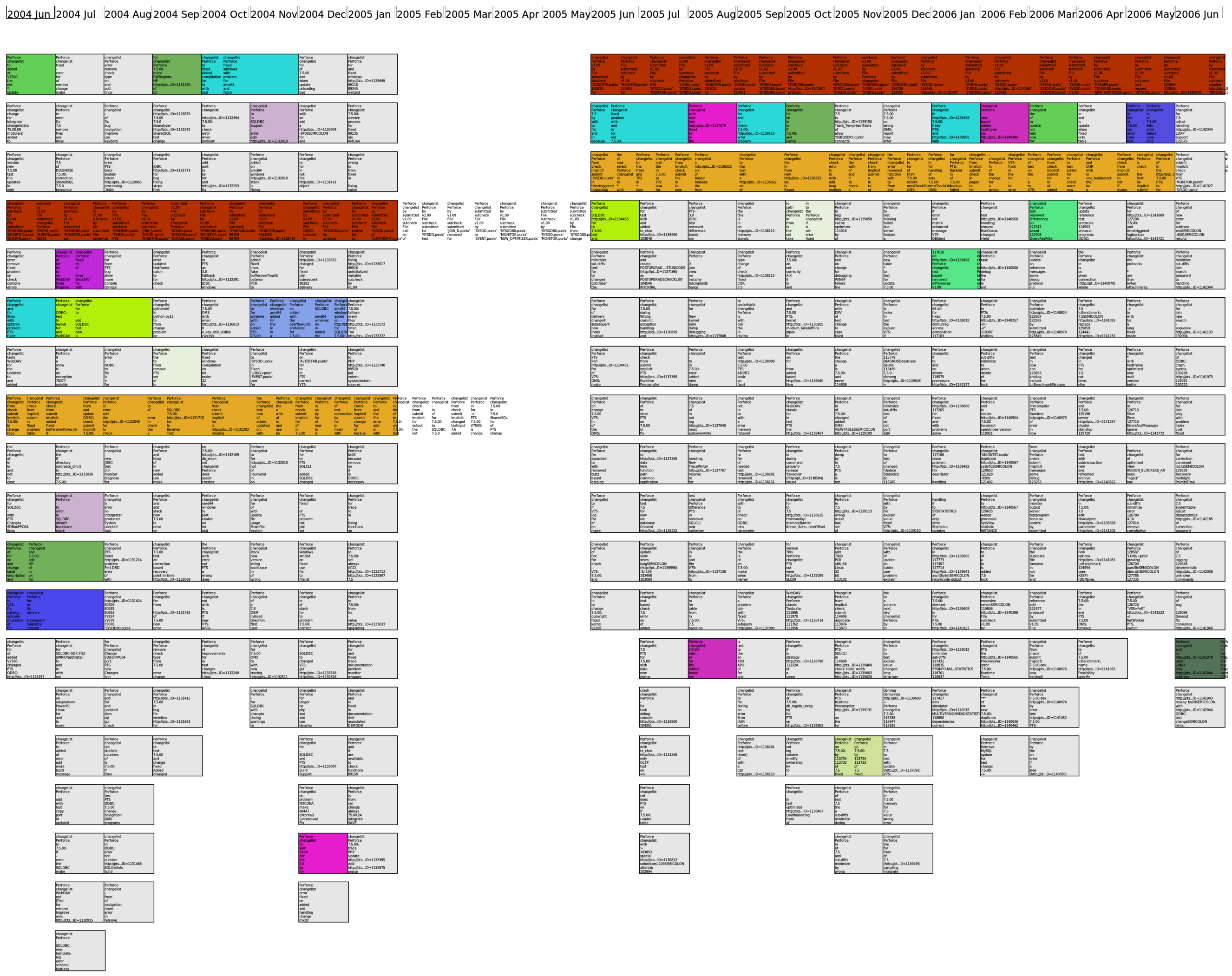

Figure 4. Compact-trend-view shows topics per month for MaxDB 7.500. The x-axis is time in months, the y-axis is used to stack topics occurring at the same time. Trends that are continuous are plotted as continuous blocks. Trends with more than one topic are colored the same unique color, while topics that do not reoccur are colored grey. The top 10 words in the topics are joined and embedded in box representing the topics. No stop words were removed. The gap in the middle is a period where no development took place (see Section 5.3).

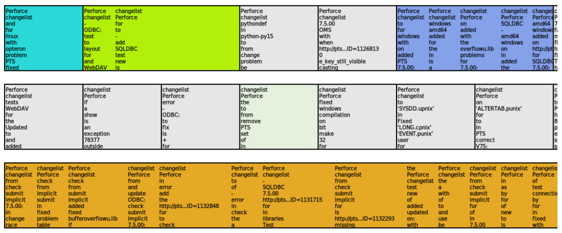

Figure 5. A zoomed-in slice of compact-trend-view of topics per month of MaxDB 7.500. The topic text is visible in each topic box. Trends are plotted across time continuously.

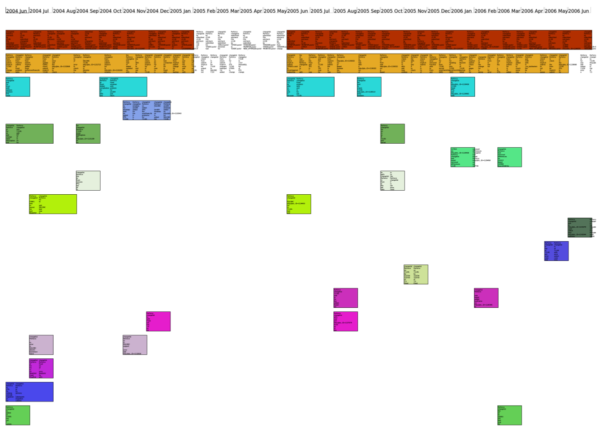

Figure 6. Trend-time-line: Trends plotted per month of MaxDB 7.500. Time in months are plotted along the x-axis, each row on the y-axis is associated with a trend ranked by size in descending order.

tries to sink 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 many 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 a trend is randomly chosen while topics that do not reoccur are colored grey. Color similarity is not meaningful. The *compact-trend-view* makes repeating continuous trends easy to pick out, although discontinuous trends are harder to spot, and provides a general summary of the topics within a project.

The *trend-time-line* (Figure 6) attempts to show a summary of trends separated from single topics. This view shows how a trend persists across time, which aids time-wise analysis of trends. The *trend-time-line* displays repeating trends more clearly by dedicating a horizontal line for trend segments belonging to one trend. Therefore 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. Thus the *trend-time-line* view is used to analyze trends across time.

The *trend-histogram* (Figure 7) attempts to show a count of how often a trend reoccurs and how many topics are related to a trend. It is meant to show the distribution of trends by their size in topics and time-span. The *trend-histogram* 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. Due to the large number of topics (approximately N topics multiplied by M periods), given the allotted space, it is often best to crop off the trends with only one topic (90% to 99% of the total topics), otherwise the tail is long. The *trend-histogram* summarizes the distribution of trends ordered by size.

All of these visualization combine to enable an analysis of trends and topics. Some views like the *compact-trend-view* enable an analysis of local topics while the *trend-histogram* and *trend-time-line* focus more on trends. We used these visualizations to analyze the database systems that we discuss in the following results section.

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. [2]; and our trend plotter, implemented in Haskell.

We applied our tools and visualizations to the repositories of three open source database systems: PostgreSQL, MaxDB, and Firebird. The total number of commits analyzed was over $66,000$.

5.1. PostgreSQL

We examined PostgreSQL's history from 1996 to 2004, which includes over $20,000$ commits. We did not find many trends with two or more topics, using a similarity of $7/10$. $7/10$ was chosen because it preserve independent topics but seemed to be a threshold value where more serious trends started to appear.

Those trends that we did find were not very large, lasting only 3 months at most. The first and second largest trends directly referenced two external developers: Dal Zotto, Dan

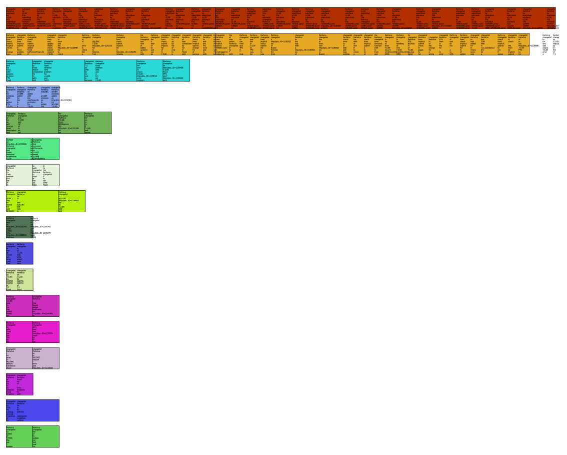

Figure 7. The top part of a trend-histogram of MaxDB 7.500, ordered by topic occurrence. X-axis determines the number of continuous months of a trend. Trends are ranked by the number of topics that a trend contains in descending order.

McGuirk. The fifth largest trend related to work by PostgreSQL developer D'Arcy. Other topics of the larger trends were changes to the "to do" list, and time string formating topics relating to time-zones.

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 $1/2$, 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 mentions the `[PATCHES]` mailing-list 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 (CVS does not record merges explicitly).

5.2. Firebird

We tracked Firebird from August 2000 to January 2006, we extracted comments from $38,000$ commits. We found that with a similarity of $7/10$ Firebird had far more continuous and recurring trends than PostgreSQL. The first large trend was discontinuous 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 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. There were a total of $8,600$ commits analyzed.

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 one unique Bug tracker URL. This pattern did not occur in the previous month. We investigated the revisions and we found that developers were referencing the bug tracker more during the last month. 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.

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, explained in Section 5.4) it is not in any of the topics. A query optimizer is an important component of a DBMS, but as we have shown it does not appear 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 found a topic with optimizer in its top 10 words. Our analysis shows that optimizer was important but it had been obscured by the global topic analysis, which used the entire history of messages, but would have been noticed using the more local topic analysis such as our windowed topic analysis, which used a smaller window of messages.

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 four different topics and one global topic. Perhaps this is because there was a cluster of *Perl* mentions during one month while the *PHP* mentions were more spread out.

Just about every topic included the words *Perforce* and *Changelist*, 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 reduced. 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 from the entire development history of a project and then tracked their relationships to documents over time, we call this *global topic analysis*.

We carried out *global topic analysis* on MaxDB 7.500 and compared this against our windowed topic analysis. To produce Figure 8, we extracted 20 topics and plotted the number of messages per month that were related to that topic. One topic would often dominate the results, as shown in the third row of Figure 8, while the other topics did not appear as 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 windowed topic analysis 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.

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 compared topics using the top 10 tokens, this approach could be throwing data away and might not be as useful as determining the actual distance between two word topic distributions.

Our choice of the number of topics and adding and removing stop words produced different results. Our choice of stop words could be biased, and could affect the results.

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 ap-

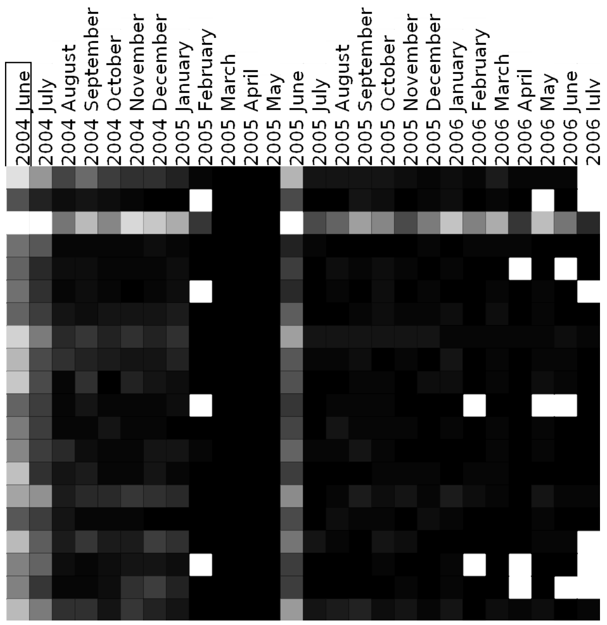

Figure 8. MaxDB 7.500 topics analyzed with global topic analysis, 20 topics (each row) and their document counts plotted over the entire development history of MaxDB 7.500 (26 months). The shade of the topic indicates the number of documents matching that topic in that month relative to the number of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

proach 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.

Windowed topic analysis demonstrated its ability to hilight local topics and identify global trends. This was shown in our case study of MaxDB 7.500. Global topic analysis missed important topics such as *ODBC* while windowed topic analysis identified them.

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 more information than the views that global analysis could show, and 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 highlights and measures the size of trends while our trend-time-line view shows how a topic reoccurs over time. These visualizations help us understand the common topics that developers focus on during development. If implemented interactively, a user could easily zoom in and query for a summary of a topic or trend. In summary, our work on windowed topic analysis shows the potential for automatically determining key topics and trends across software development projects .

7.1. Future Work

We wish to extend this study by further exploring parameter choices and their effects in terms of window overlap size and number of topics. Another 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.

Acknowledgements: We would like to thank Tudor Girba and Neil Ernst for their feedback and ideas.

References

[1] T. Akira. Cvssuck - inefficient CVS repository grabber. http://cvs.m17n.org/ akr/cvssuck/.

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

[3] K. Delac, M. Grgic, and S. Grgic. Independent comparative study of PCA, ICA, and LDA on the FERET data set. *IJIST*, 15(5):252–260, 2006.

[4] D. M. German, A. Hindle, and N. Jordan. Visualizing the evolution of software using softchange. In *SEKE*, 2004.

[5] S. Grant, J. R. Cordy, and D. Skillicorn. Automated concept location using independent component analysis. In *WCRE*, 2008.

[6] A. Hindle, M. Godfrey, and R. Holt. Release Pattern Discovery via Partitioning: Methodology and Case Study. In *MSR*, 2007.

[7] A. Ko, B. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. 2006.

[8] A. Kuhn, S. Ducasse, and T. Girba. Enriching reverse engineering with semantic clustering. 2005.

[9] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining concepts from code with probabilistic topic models. In *ASE '07*. ACM, 2007.

[10] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining eclipse developer contributions via author-topic models. 2007.

[11] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic. An information retrieval approach to concept location in source code. 2004.

[12] D. Poshyvanyk and A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. 2007.

[13] L. H. E. Stacy K. Lukins, Nicholas A. Kraft. Source code retrieval for bug localization using latent dirichlet allocation. In *WCRE*, 2008.

[14] A. Tridgell, P. Mackerras, and W. Davison. Rsync. http://www.samba.org/rsync/.