Cliff Walls: An Analysis of Monolithic Commits Using Latent Dirichlet Allocation

Landon Pratt, Alexander Maclean, Charles Knutson, Eric Ringger

Cliff Walls: An Analysis of Monolithic Commits Using Latent Dirichlet Allocation

Landon J. Pratt, Alexander C. MacLean, Charles D. Knutson, and Eric K. Ringger

Computer Science Department, Brigham Young University, Provo, Utah
landonjpratt@byu.edu, amaclean@byu.edu, knutson@cs.byu.edu, ringger@cs.byu.edu

Abstract. Artifact-based research provides a mechanism whereby researchers may study the creation of software yet avoid many of the difficulties of direct observation and experimentation. However, there are still many challenges that can affect the quality of artifact-based studies, especially those studies examining software evolution. Large commits, which we refer to as "Cliff Walls," are one significant threat to studies of software evolution because they do not appear to represent incremental development. We used Latent Dirichlet Allocation to extract topics from over 2 million commit log messages, taken from 10,000 SourceForge projects. The topics generated through this method were then analyzed to determine the causes of over 9,000 of the largest commits. We found that branch merges, code imports, and auto-generated documentation were significant causes of large commits. We also found that corrective maintenance tasks, such as bug fixes, did not play a significant role in the creation of large commits.

1 Introduction

Artifact-based software engineering research may in some respects be compared to archaeology, a field that has been defined as "the study of the human past, through the material traces of it that have survived"[2]. Much like archaeologists, empirical software engineering researchers often seek to understand people. The software engineering researcher, while not isolated from a target population by eons, faces other obstacles that often make direct observation impossible. Many organizations are loath to allow researchers through their gates, in an effort to protect trade secrets or merely to hide shortcomings. Even in cases where researchers are allowed to directly observe engineers building software, the Hawthorne effect threatens the validity of such observations. Time investment and organizational complexity are also issues that pose problems in software engineering research. Direct observation requires significant time investment, making it impossible for a single researcher to observe everything that takes place within a given software organization.

As a result of these barriers, software researchers, like their archaeologist counterparts, take advantage of artifacts—work products left behind in software

project burial grounds. Artifacts are collected after the fact, minimizing the confounding influence of the presence of a researcher. Artifacts also help researchers deal with the requirements of studying complex organizations. By leveraging artifacts of the software process, researchers are able to study thousands of pieces of software in a relatively short period of time, an otherwise impossible task.

The open source movement is particularly important to software engineering research, since project artifacts, such as source code, revision control histories, and message boards, are openly available to software archaeologists. This makes open source software an ideal target for researchers with a desire to understand how software is built.

1.1 Threats to Artifact-based Research

Unfortunately, the study of artifacts in software engineering is not all sunshine and double rainbows; serious challenges threaten the results of artifact-based research involving open source software projects (such as those hosted on SourceForge). Since artifact data is examined separate from its original development context, identifying the development artifacts actually recorded in the data can be difficult. It is challenging enough to ensure that measurements taken for a specific purpose actually measure what they claim to measure [5]. It is all the more difficult (and necessary), therefore, to validate artifact data, which is generally collected without a targeted purpose.

Many phenomena can easily be overlooked by software archaeologists, such as auto-generated code, the presence of non-source code files in version control, and the side effects of branching and merging. Understanding the limitations of artifact data represents an important step toward validating the results of numerous studies (for example, [3, 6, 9, 13, 18, 20, 22]). Despite on-going efforts to identify and mitigate the limitations of artifact-based research, new threats are constantly emerging.

The focus of this paper is one such threat—the presence of monolithic commits in open source repositories. A close look at the version control history of many projects in SourceForge reveals some worrisome anomalies. Massive commits, which we refer to as "Cliff Walls," appear with alarming frequency. One investigation of Cliff Walls in SourceForge found that out of almost 10,000 projects studied, half contained a single commit that represented 30% or more of the entire project size [16]. These Cliff Walls indicate periods of unexplained acceleration in development, threatening some common assumptions of artifact-based research, especially for studies of project evolution. Cliff Walls thwart attempts to tie authors to contributions, mask true development history, and taint studies of project evolution. We must better understand Cliff Walls if we are to paint an accurate picture of software evolution through the use of artifacts.

2 Cliff Walls

In [16] we introduced the concept of "Cliff Walls" as unexpected increases in version control activity over short periods of time (see Fig. 1). In the most extreme

cases, Cliff Walls represent millions of lines of source code contributed to the repository in less than a day. Such activity is problematic for researchers, especially those investigating the evolution of software, because such sudden surges of source code commits cannot possibly be the result of common incremental development activities. Even a team of "supercoders" would be hard-pressed to produce such massive quantities of code in a short period of time, all the more impossible for the single "author" to which a version control commit is attributed. We are forced to ask what these "Cliff Walls" truly represent.

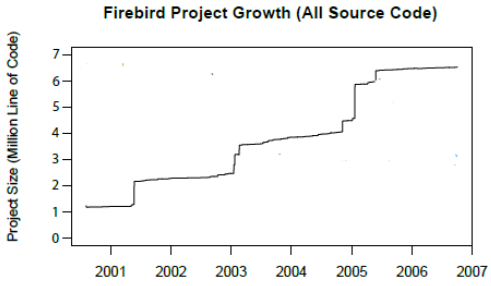

Fig. 1. Cliff Walls in the Firebird Project

2.1 Definitions

When examining contributions to version control systems, it is logical to discuss them in the context of "commits." However, some researchers have defined "commit size" in terms of the number of files changed in a commit [12, 11, 10], while others treat the size of a commit as a function of the number of LOC added, removed or modified [1].

Within the context of this study, a *commit* refers to the set of files and/or changes to files submitted simultaneously to a version control system by an individual developer. Since we are primarily concerned with the growth of projects over time, *Commit size* is defined as the total number of lines added or removed from all source code files in a given commit, which allows us to perceive code growth (or shrinkage) within a project. For example, if an existing line in one file is modified, and no other changes are made, the size of the resulting commit would be zero LOC. In this study, we include code comments in the measurement of commit size, allowing us to detect code growth attributable to the insertion of comments within source files.

We use the term *Cliff Wall* to describe any single commit with size greater than 10,000 lines of code (LOC). We believe this threshold to be sufficiently high that it avoids capturing most incremental development contributions for individual authors. While it is possible that an extremely productive developer could produce 10,000 lines of code in a period of days or weeks, it is unlikely.

Additionally, the methods employed in this study should allow us to identify instances where the threshold fails.

2.2 Commit Taxonomies

The ability to distinguish between different types of cliff walls is critical for many artifact-based studies. For example, a researcher attempting to measure developer productivity on a project would likely want to be able to distinguish between large commits caused by auto-generated code and branch merges, as they must be handled differently when calculating code contributions.

In [12], the authors present a taxonomy used to manually categorize large commits on a number of open source software projects. Hattori and Lanza also present a method for the automatic classification of commits of all sizes in open source software projects [10]. The taxonomies developed in these studies focus on classifying the type of change made (for example, development vs. maintenance). Both these studies concluded that corrective maintenance activities, such as bug fixes, are rarely the topic of large commits. These studies also found that a significant portion of large commits were dedicated to "implementation activities." The authors consider all source code files added to the repository to be the result of implementation activities.

In [1], the authors categorize commits from open source projects into three groups, using a rough heuristic based on commit size. Their study makes no attempt to analyze commit log messages. Rather, they divide each commit into one of three groups, based on the commit size: 1) single individual developer contributions, 2) aggregate developer contributions, and 3) component or repository refactoring or consolidations.

In our study, we seek to identify specific behaviors that play a significant role in the creation of monolithic commits. In the next section, we discuss the methods that we will apply in our search for the causes of Cliff Walls.

3 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is an unsupervised, hierarchical Bayesian topic model for analyzing natural language document content that clusters the terms in a corpus into topics [4]. In LDA, the model represents the assumption that documents are a mixture of these topics and that each topic is a probability distribution over words. In short, LDA allows us, without any knowledge engineering, to discover the topical content of a large collection of documents in an unsupervised fashion. Given a corpus of documents such as our commit log messages, inference techniques such as Gibbs sampling or Variational Bayes can be used to find a set of topics that occur in that corpus, along with a topic assignment to each word token in the corpus.

A common technique for visualizing the topics found using LDA is to list the top n most probable terms in each topic. For example, one of the topics we found using LDA that concerns bug correction, consisted of the following terms:

fix bug fixes crash sf problems small quick closes blah deleting weapon decoder lost hang weapons delphi noted led

When brevity is required, it is also common to refer to a topic by just the first two or three most probable terms, sometimes in hyphenated form. Using this notation, the above topic could be referred to as the "fix-bug" topic. Both of these methods for indicating a particular topic (word list and first words) are used throughout this paper.

Because LDA also assigns a topic to each individual token within a commit message, a single commit can, and usually does, contain many topics. This is one benefit of using LDA: each "document" may belong to multiple classes. Such is not the case with many of the supervised classification methods, which typically assign each document to a single class. In the case of commit log messages, allowing for multiple topics in a single message allows us to conduct a more fined-grained analysis.

The approach taken by LDA is an attractive alternative to other classification and clustering methods. Supervised classification methods require a training set of "tagged" data, for which the appropriate class for each instance has previously been specified. In many situations this tagged data does not exist, requiring the researcher to manually assign tags to some portion of the data. This is a tedious, time-consuming, and potentially biased process which can be avoided through the use of unsupervised methods such as LDA. Unsupervised methods may also compensate for a measure of "short-sightedness" by the researcher. In supervised methods, since the classes must be predefined by the researcher, it is possible to degrade the usefulness of the model through the omission of important classes, or the inclusion of irrelevant classes. Unsupervised methods avoid some of these errors since no predefined clusters are specified, allowing the data to better "speak for itself." For our analysis, we used the open source tool "MALLET" which includes an implementation of LDA [17].

4 Methods

Our data set consists of the version control logs of almost 10,000 projects from SourceForge, acquired in late 2006. This data set has been previously used in a number of studies [6, 7, 8, 13, 14, 15, 16]. The logs for all projects in our data set were extracted from the CVS version control system. In creating the data set, the original authors filtered projects based on development stage; only projects labeled as Production/Stable or Maintenance were included in the data set. For further description of the data, see [6]. In calculating commit size, we excluded non-source code files, based on file extension. Over 30 "languages" were represented in the final data set, including Java, C, C++, PHP, HTML, SQL, Perl and Python.

Because CVS logs do not maintain any concept of an atomic multi-file "commit" it was necessary to infer individual commits. We utilized the "Sliding Time Window" method introduced by Zimmerman and Weißgerber [23]. This resulted

in a set of almost 2.5 million individual commits, extracted from over 26 million file revisions. Applying our pre-defined threshold of 10,000 LOC yielded over 10,000 Cliff Walls. We also found that a number of the commits contained log messages that were uninformative. Commits with empty log messages or with **"empty log message"** were removed from the data to prevent degradation in the quality of topics identified. The resulting set contained 2,333,675 commits, with 9,615 Cliff Walls. We later removed other uninformative commits (see discussion in Sec. 6), ultimately resulting in the exclusion of 6.6% of commits in our data set due to log messages that conveyed no information about the development activities they represent. A disproportionate number of the commits removed were Cliff Walls (an ultimate exclusion of 14.8% of all Cliff Walls). Additionally, very common English adverbs, conjunctions, pronouns and prepositions belonging to our "stop-word" list were removed from the commit messages in order to ensure the identification of meaningful topics.

The LDA algorithm, as implemented in MALLET, requires three input parameters: the number of topics to produce in its analysis, the number of iterations, and the length of the burn-in period. In our study, we elected to identify 150 topics with MALLET. The authors Hindle and German identified 28 "types of change" for the commits classified as a part of their taxonomy [12]. Hattori and Lanza, in their study of commit messages, identified 64 "keywords" that were used to classify commits [10]. These prior results gave us reason to believe that 150 topics would be a sufficient number to capture the motivations behind the commits in our data set, with an appropriate level of detail.[1]

As one of the steps to understanding the Cliff Wall phenomenon, we compare the most prevalent topics found in Cliff Wall commits to those found in the entire body of commits. Instead of running LDA separately on the two subsets of our data, we run LDA once on all of the data and then filter the results to gain a picture of Cliff Walls in contrast to All Commits. This approach ensures that the topics found are consistent across both groups, which helps yield a meaningful comparison.

5 Analysis & Discussion

Figure 2 provides a first glance at some of the variation exhibited by Cliff Walls. In these graphs, each horizontal bar represents one of the 150 topics generated. The thickness of each bar represents the proportion of tokens in the entire corpus of commits that were assigned to that topic. Commit log messages are fairly evenly distributed over the topics for the general population of commits.

[1]The other two parameters, number of iterations and length of burn-in period, are required by the Gibbs Sampling inference method employed by MALLET. We refer the reader to [21] for a description of LDA as it is implemented within MALLET, including a description of Gibbs Sampling. For these two parameters we used the default values provided by MALLET; 1,000 iterations with 200 dedicated to burn-in. Further work should investigate the possibility of more appropriate values for all three parameters, as discussed in Sec. 6.

However, a small number of topics are considerably more prevalent in the large commits. Tables 1 and 2 list the 15 most prevalent topics for all commits and Cliff Wall commits.

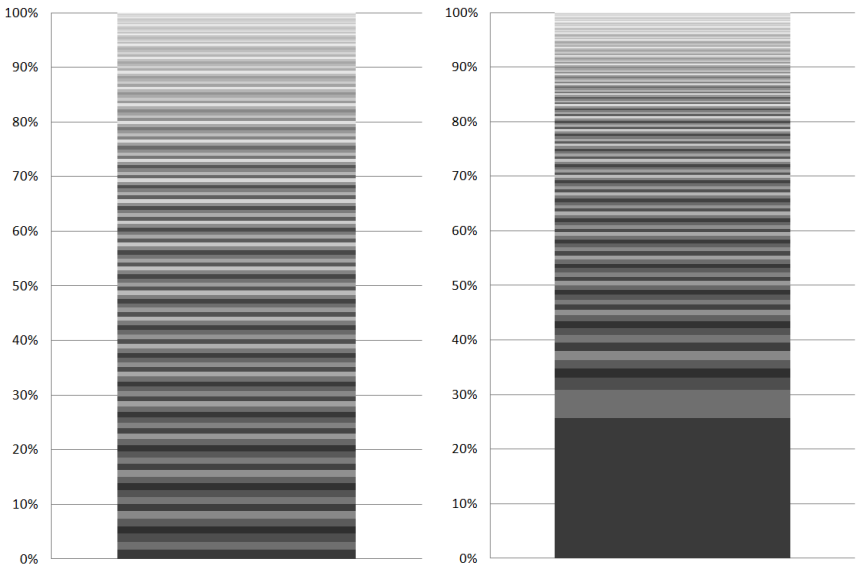

Fig. 2. Topic distribution for All Commits (left) and Cliff Wall Commits (right)

Similarly, the tag clouds in Figs. 3, 4, and 5 begin to give us an idea of the most common topics for our two groups of interest.[2] Each tag in the cloud represents a topic that has been summarized using the "first words" method described in Sec. 3. Like a stratum from Fig. 2, each tag is sized based on the proportion of tokens belonging to that topic. Thus, the largest tag in Fig. 4, "initial-import," is also the largest stratum in the bar chart for Cliff Walls. Tag position and color do not convey any additional information. Figure 5 is an alternate view of Fig. 4, with the dominant "initial-import" topic removed to improve readability. These images provide an overall view of the topics and their proportions for the two groups of interest. We next discuss some of the most prevalent topics and their interpretations.

One of the goals of this paper is to compare the topics produced by LDA with previously hypothesized causes of Cliff Walls. We examine each of these causes to see if LDA is able to identify them as prominent features of large commits. We also examine some of the causes for Cliff Walls that were previously overlooked, but were consequently suggested by LDA. In pursuit of these goals, we have found two views of the data that provide insight into the causes of Cliff Walls:

[2] All tag cloud images were generated using Wordle (www.wordle.net).

Fig. 3. Topic Tag Cloud: All Commits

Fig. 4. Topic Tag Cloud: Cliff Walls

Fig. 5. Topic Tag Cloud: Cliff Walls ("initial-import" excluded)

Overall Topic Proportion and Topic Relative Rank. We discuss these in the next two subsections.

5.1 Overall Topic Proportion

Tables 1 and 2 display the 15 most prevalent topics in the Cliff Wall and All Commits groups, as determined by proportion of tokens belonging to that topic. For example, the topic "version-release" in Tbl. 1 has a proportion of 1.75%, suggesting that 1.75% of all the words in all of the commit messages in our data set were assigned to this topic by the LDA algorithm. In other words, these tables list the topics most frequently discussed in commit log messages belonging to our two groups of interest. We refer back to these tables frequently throughout this section as we discuss the various causes of Cliff Walls.

5.2 Topic Relative Rank

Additional insight into Cliff Walls can be gained through the use of another simple metric. Within each of our two groups, "All Commits" and "Cliff Walls," each topic can be ranked based on its proportion within the group. The difference between a topic's ranking in the two groups is a good indicator of the prevalence of a given topic relative to other topics.

For example, in Tbl. 1 we see that "initial-import" is the 5th ranked topic. In Table 2, the same topic is ranked 1st, for a rank difference of +4. In contrast, as we see from Tbl. 3, the topic "cvs-sync" holds ranks of 101 and 4, resulting in a rank difference of +97. In essence this means that, relative to other topics, "cvs-sync" is discussed more frequently within Cliff Wall commits than it is for the general population of commits.

It is important to note that the difference between topic ranks is not synonymous with a similar difference in proportion. The difference between proportions for the "initial-import" topic is an approximately 25% increase for the Cliff Walls group. This is a very large change in proportion which results in a relatively small

Table 1. Top 15 Topics for All Commits

#	Proportion	Key Terms				
1	1.75%	version	release	updated	update	final
2	1.43%	file	branch	initially	added	java
3	1.42%	fixes	minor	small	cleanups	updates
4	1.38%	data	minor	updates	fixes	bugfixes
5	1.38%	initial	import	commit	checkin	revision
6	1.36%	added	comments	comment	documentation	javadoc
7	1.29%	fixed	bug	bugs	incorrect	couple
8	1.28%	added	support	info	extended	basic
9	1.28%	error	message	errors	handling	checking
10	1.26%	removed	code	template	commented	unnecessary
11	1.25%	fix	bug	fixing	small	bugs
12	1.18%	fix	typo	corrected	correct	errors
13	1.17%	fixed	bug	wrong	crash	introduced
14	1.17%	page	link	updated	links	url
15	1.14%	code	cleanup	source	clean	cleaned

Table 2. Top 15 Topics for Cliff Walls

#	Proportion	Key Terms				
1	25.74%	initial	import	commit	checkin	revision
2	5.11%	version	release	updated	update	final
3	2.30%	removed	deprecated	sources	constant	imported
4	1.63%	cvs	sync	tabs	real	converted
5	1.60%	error	message	errors	handling	checking
6	1.58%	message	project	messages	error	testing
7	1.51%	merged	merge	head	merging	main
8	1.47%	code	cleanup	source	clean	cleaned
9	1.29%	files	added	dialogs	library	directories
10	1.21%	directory	moved	common	dir	structure
11	1.17%	xml	updated	api	latest	version
12	0.98%	update	added	format	updating	creation
13	0.93%	script	makefile	install	configure	sh
14	0.89%	added	comments	comment	documentation	javadoc
15	0.89%	double	beta	v1	v2	values

Table 3. Largest "Positive" Rank Differences

	All Rank	Cliff Wall Rank	Key Terms			
+97	101	4	cvs	sync	update	repository
+97	112	15	beta	v1	v2	v3
+93	131	38	org	cvs	synchronized	packages
+91	98	7	merged	merged	trunk	stable
+91	100	9	files	added	directories	library
+89	99	10	directory	moved	structure	location
+88	142	54	cc	net	users	sourceforge
+80	128	48	module	python	py	libsrc
+73	119	46	system	specific	platform	devel
+71	133	62	http	www	urls	net
+71	105	34	web	site	resource	component
+67	109	42	php	index	http	forum

Table 4. Largest "Negative" Rank Differences

	All Rank	Cliff Wall Rank	Key Terms			
-120	13	133	fixed	bug	wrong	crash
-118	12	130	fix	typo	corrected	correct
-116	7	123	fixed	bug	bugs	incorrect
-105	44	149	button	selection	tab	dialog
-90	23	113	fixed	problems	issue	bad
-89	30	119	string	return	null	true
-80	51	131	variable	global	unused	define
-77	26	103	output	debug	print	messages
-76	49	125	problem	fixed	patch	solution
-75	36	111	size	buffer	limit	bytes
-75	54	129	function	calls	static	inline
-74	3	77	fixes	minor	cleanups	cosmetic

difference in rank. It is even possible, given the distinct distributions of our two groups (see Fig. 2) that a *negative* change in proportion could still result in a *positive* rank difference.

5.3 Code Imports

A "Code Import" occurs when a significant amount of source code is added to the repository. Code Imports differ from "Off-line Development" in that the code added was not developed as part of the project of interest, but instead originated from some other project. The most common example of a code import is probably the addition of the source code for an externally developed library.

We found a good deal of evidence that Off-line Development is a significant cause of Cliff Walls. As shown in Tbl. 2, a few prominent topics (particularly 1,

4 and 9) deal with the first-time addition of files to the repository. The addition of files alone, however, does not indicate a Code Import. Table 1 indicates that topic 1 is quite prominent for all commits, because files are constantly being added to version control systems. In the case of Cliff Walls, however, the size of the commit gives us good reason to believe that the files added contain a great deal of code, and therefore do not represent files added as part of incremental development.

Also, a few of the topics do provide additional evidence of Code Imports. Topic 9 contains the term "library" which indicates that this topic relates to the addition of library files to version control. Similarly, the topic "module-python" appears in Tbl. 3 as a topic with a much higher relative rank for Cliff Wall commits. Examination of log messages for which this topic had high proportion yielded messages such as "bundle all of jython 2.1 with marathon so all python standard library is available" and "Add the Sandia RTSCTS module to the code base." These messages are indicative of Code Imports.

5.4 Off-line development

In this paper, we use the term "Off-line Development" to refer to large quantities of code that were developed as part of a project, but for which we have no record. This may be code that was developed without the benefit of version control, or that was developed in a separate repository and then added to the SourceForge CVS repository in a monolithic chunk.

Much of the evidence for Off-line Development is similar to that of Code Imports. Many of the same topics that may refer to code imports ("initial-import," "cvs-sync," and "files-added") could equally be attributed to Off-line Development. Thus it is difficult to distinguish between the true source of many large commits, because it is hard to tell if the files added were developed in conjunction with the current project or separately. Further investigation is required to elucidate the differences between Cliff Walls attributable to Code Import and those due to Off-line Development.

5.5 Branching & Merging

Merging is a major factor in the creation of Cliff Walls. The 7th ranked topic for Cliff Walls is a topic dealing with the merging of a branch in the repository. This same topic also appears as one of the largest positive rank differences in Tbl. 3. This indicates that not only are merges a significant factor behind the creation of Cliff Walls but also that the merge topic is significantly more prevalent within Cliff Walls than it is for All Commits. In contrast, the "initial-import" topic is one of the highest ranked topics in both groups.

5.6 Auto-Generated Code

Topics pertaining to Auto-generated Code are a bit more difficult to identify. The topic "target-generated" appears to capture auto-generated code quite well:

target generated rules rule generate mark reports make generation targets automatically linked policy libraries based generator jam dependencies building

Surprisingly, this topic is of relatively little importance, with rank 67 (Cliff Walls) and 113 (All Commits). Such low ranks would seem to indicate that Auto-generated code does not play a significant role in the explanation of Cliff Walls. We did, however, find another example that suggests that Auto-generated Code may be a more significant factor. We were surprised to find that the topic "added-comments" was the 14th ranked topic for Cliff Walls (see Tbl. 2). Non-source code files had been excluded from our study, and code commenting seemed an unlikely cause for commits on the magnitude of 10,000 lines or greater. Upon appeal to the commit log messages, we found a large number of messages containing text such as "Add generated documentation files," "Documentation generated from javadoc," and "Updated documentation using new doxygen configuration."

Further examination revealed that, at least in the cases mentioned above, these commits consist almost entirely of HTML files. The above comments contain 81, 120, and 448 HTML files, respectively. This suggests that large, comment or documentation related commits may be the result of auto-generated HTML files from documentation systems such as javadoc and doxygen.

It is possible that there may be other significant sources of Auto-generated Code expressed in the topics obtained from LDA. In the above case, the tools that generated the "code" were more effective identifiers of Auto-generated code than were the terms "automatically" and "generated." Further investigation is required to determine whether other such cases exist.

5.7 Other Findings

As we hoped, the application of LDA to this problem suggested some potential Cliff Wall causes we had not forseen. Additionally, a few interesting observations served to confirm some of our suspicions about Cliff Walls. In this section we discuss some of these findings.

One discovery of note was the importance of activities related to project releases and new versions. The 2nd most prevalent topic discussed in Cliff Wall log messages is "version-release." Topics 11 and 15 in Tbl. 2 also deal primarily with project releases and versioning, with prevalent terms such as "latest," "version," "beta," "v1" and "v2." It is difficult to tell exactly what is occurring with these commits; most provide little information other than a version number. We suspect that many of these may be the result of merges, and further investigation may determine the true cause.

We were able to gain some understanding of topics which were infrequently discussed in the log messages of Cliff Wall commits. Table 4 shows some of the topics for which the topic rank dropped significantly for Cliff Wall commits. This drop would indicate topics that were discussed much less frequently for Cliff Walls than All Commits, when compared to all other topics. Some of the trends in the table include topics discussing corrective maintenance ("fixed-bug,"

"fix-typo," 'fixes-minor," "output-debug"), gui tweaks ("button-selection"), and minor implementation details ("string-return," "variable-global," "size-buffer"). It is not surprising that these topics do not significantly occur in the log messages of large commits, but these trends lend credibility to our results.

Table 3 provides another interesting insight. We observe that two topics appear to deal with web technologies: "http-www" and "php-index." In many cases, we found that these topics indicated the presence of a URL in the log message. It is intriguing that this topic surpassed so many others in the Cliff Walls category. We believe that these URLs could convey valuable information about the commit, and may help to identify library code that is being imported, or the location of an external version control repository utilized by the project.

6 Threats

Some of the most significant threats to our results arise from the data set employed. As previously stated, the data was gathered in late 2006, and is now relatively old. It is possible that the results that we have found do not correspond to the current state of projects in SourceForge. This study is also limited to projects using the CVS version control system. According to our estimates, almost all new projects in SourceForge are now using Subversion instead of CVS. While the two technologies are similar in many ways, it is possible that our analysis would produce different results if conducted using data from Subversion logs.

It should be noted that when the original data was gathered, projects were filtered to include only those projects listed as "Production," "Stable," or "Maintenance," in an effort to limit the data set to include only "successful" projects [6]. As a result, when we talk about topics across "All Commits," we are actually unable to generalize to the entire population of projects in SourceForge. This is significant, because as one estimate found, only about 12% of projects in SourceForge were being actively developed [19]. It is possible, even likely, that a similar analysis, not limited to "Production/Stable/Maintenance" projects would produce different results. However, we do not feel that the depiction of Cliff Walls would change dramatically, as we presume they are rare in defunct projects.

Another significant threat to the validity of our results is the presence of "low quality" topics. We found two types of low quality topics in our results: topics with contradictory terms and topics generated from dirty data. One example of a topic with contradictory terms is the "removed-deprecated" topic ranked 3rd in Tbl. 2. This topic contains the contradictory terms "removed," and "imported" as important terms in the topic. This leads to a topic that is difficult to interpret, as the "meaning" of the topic can vary based on the document in which it is present. To better understand this topic we examined the log messages of 233 Cliff Walls containing the topic. Of those 233, the term "removed" occurred in only 1 message, while "imported" occurred in 154. Obviously, for large commits, "imported" is a much more appropriate description of log messages with this topic.

We found two low quality topics resulting from dirty data in our results. The topics "error-message" and "message-project," the 5th and 6th most prevalent topics for Cliff Walls, are also misleading. We looked at 286 Cliff Wall log messages containing at least one of these two topics, and found that 211 (74%) of them contained only the message "no message." These commits should have been removed from the data set prior to the analysis. Exclusion of these commits would result in a data set containing 2,301,620 commits, with 9,199 Cliff Walls, a minor decrease in size. We do not feel that this issue greatly affected the outcomes of this study. However, these topics possibly prevented more appropriate topics from being considered.

In order to improve the results of future studies applying LDA to the Cliff Walls problem, greater effort should be made in LDA model selection. The three input parameters that we were required to specify (number of topics, number of iterations, and burn-in period) could likely be tuned to produce higher quality topics. In particular, this may help us to avoid the issue of topics with contradictory terms.

7 Conclusions

We are excited by the promise that LDA shows for automated analysis of large commits. Through the use of tag clouds and other views of the data, we have been able to gain an insightful picture into the causes behind Cliff Walls. We found that in most cases, our suspicions of Cliff Walls were confirmed. We found significant evidence that library imports, externally developed code, and merges were the subjects of topics frequently discussed in log messages of large commits. We also found evidence that auto-generated code can, in some cases, result in the creation of cliff walls. LDA also helped us to confirm that maintenance tasks, such as bug fixes, do not occur in large commits with much frequency. These conclusions agree with previous studies on the causes of large commits [12, 10].

We found that it was difficult to use commit log messages to distinguish library code imports from imports of large amounts of project code. However, in some cases we are able to identify library imports. We also hope that, in the future, the URLs included in some Cliff Wall commit messages may be used to identify other instances of library code imports.

We believe that LDA is a welcome alternative to many of the methods that have previously been used for classification of commit log messages. While we invested a great deal of time manually interpreting the results produced by LDA, we were able to avoid the tedium of data tagging required by most supervised classification tasks.

8 Future Work

The role of large commits in software evolution is still largely unclear. In this study, we have examined the causes of Cliff Walls for a particular subset of all

software projects—those that are relatively successful, are hosted on SourceForge, and that use CVS for version control. In order to better understand Cliff Walls, we need to build upon this subset. First, research should consider investigating Cliff Walls as they occur in other version control systems. CVS is no longer the most significant version control system, since others, such as Subversion and GIT have risen to take its place.

SourceForge is only one of many environments in which open source software is developed. There are various open source forges and foundations, each with its own tools, communities, policies, and practices that influence the software development that occurs therein. It is possible that some of these other environments may prove more welcoming to those interested in studying the evolution of software. An effort should be made to characterize and compare the Cliff Walls that exist in other open source development communities, such as the Apache and Eclipse Foundations, RubyForge, and GITHub, to name a few. Of course the study of large commits should not be limited to only open source organizations, but should be investigated wherever possible.

More information may also be gained through a more in-depth analysis of the Cliff Walls themselves. The largest commit in our data set was over 13 million lines of code. In contrast, the smallest Cliff Wall contained 10,001 LOC. In this study, both of these commits, as well as everything in between, were lumped into the same class: Cliff Walls. It is likely that such a large level of granularity hides much that can be learned about the causes of Cliff Walls. We believe that there are opportunities to better understand this phenomenon by examining more closely the causes behind Cliff Walls of differing magnitudes.

Acknowledgements

The authors would like to thank Dan Walker of the BYU Natural Language Processing Lab for his willingness to provide insight and guidance on the methods used in this paper.

References

[1] O. Arafat and D. Riehle. The Commit Size Distribution of Open Source Software. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–8. IEEE, 2009.

[2] P. Bahn, P.G. Bahn, and B. Tidy. *Archaeology: a very short introduction*. Oxford University Press, USA, 2000.

[3] Christian Bird, Alex Gourley, Prem Devanbu, Anand Swaminathan, and Greta Hsu. Open borders? immigration in open source projects. *Mining Software Repositories, International Workshop on*, 0:6, 2007.

[4] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[5] L.C. Briand, S. Morasca, and V.R. Basili. Defining and validating measures for object-based high-level design. *IEEE Transactions on Software Engineering*, 25(5):722–743, Sep/Oct 1999.

[6] Daniel P. Delorey, Charles D. Knutson, and Scott Chun. Do programming languages affect productivity? a case study using data from open source projects. In *1st International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS '07)*, May 2007.

[7] Daniel P. Delorey, Charles D. Knutson, and Christophe Giraud-Carrier. Programming language trends in open source development: An evaluation using data from all production phase sourceforge projects. In *2nd International Workshop on Public Data about Software Development (WoPDaSD '07)*, June 2007.

[8] Daniel P. Delorey, Charles D. Knutson, and Alex MacLean. Studying production phase sourceforge projects: A case study using cvs2mysql and sfra+. In *Second International Workshop on Public Data about Software Development (WoPDaSD '07)*, June 2007.

[9] Ahmed E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*, pages 78–88, New York, NY, USA, 2009. ACM.

[10] L.P. Hattori and M. Lanza. On the nature of commits. In *Automated Software Engineering-Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on*, pages 63–71. IEEE, 2008.

[11] A. Hindle, DM German, MW Godfrey, and RC Holt. Automatic classication of large changes into maintenance categories. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*, pages 30–39. IEEE, 2009.

[12] A. Hindle, D.M. German, and R. Holt. What do large commits tell us?: a taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 99–108. ACM, 2008.

[13] Jonathan L. Krein, Alexander C. MacLean, Daniel P. Delorey, Charles D. Knutson, and Dennis L. Eggett. Language entropy: A metric for characterization of author programming language distribution. *4th Workshop on Public Data about Software Development*, 2009.

[14] Jonathan L. Krein, Alexander C. MacLean, Daniel P. Delorey, Charles D. Knutson, and Dennis L. Eggett. Impact of programming language fragmentation on developer productivity: a sourceforge empirical study. In *International Journal of Open Source Software and Processes (IJOSSP)*, Publication Pending.

[15] Alexander C. MacLean, Landon J. Pratt, Jonathan L. Krein, and Charles D. Knutson. Threats to validity in analysis of language fragmentation on sourceforge data. In *Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research (RESER '10)*, page 6, May 2010.

[16] Alexander C. MacLean, Landon J. Pratt, Jonathan L. Krein, and Charles D. Knutson. Trends that affect temporal analysis using sourceforge data. In *Proceedings of the 5th International Workshop on Public Data about Software Development (WoPDaSD '10)*, page 6, June 2010.

[17] Andrew Kachites McCallum. MALLET: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[18] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

[19] A. Rainer and S. Gale. Evaluating the Quality and Quantity of Data on Open Source Software Projects. 2005.

[20] Alexander Tarvo. Mining software history to improve software maintenance quality: A case study. *IEEE Software*, 26(1):34–40, 2009.

[21] H. Wallach, D. Mimno, and A. McCallum. Rethinking LDA: Why priors matter. *Advances in Neural Information Processing Systems*, 22:1973–1981, 2009.

[22] Jin Xu, Yongqin Gao, Scott Christley, and Gregory Madey. A topological analysis of the open souce software development community. *HICSS '05: Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 7, 2005.

[23] T. Zimmermann and P. Weißgerber. Preprocessing CVS data for fine-grained analysis. In *Proceedings 1st International Workshop on Mining Software Repositories (MSR 2004)*, pages 2–6. Citeseer.