What's in a name? On the automated topic naming of software maintenance activities

Abram Hindle
David Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, CANADA
ahindle@uwaterloo.ca

Neil A. Ernst
Department of Computer Science
University of Toronto
Toronto, Ontario, CANADA
nernst@cs.toronto.edu

Michael W. Godfrey
David Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, CANADA
migod@uwaterloo.ca

Richard C. Holt
David Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, CANADA
holt@uwaterloo.ca

John Mylopoulos
Department of Computer Science
University of Toronto
Toronto, Ontario, CANADA
jm@cs.toronto.edu

ABSTRACT

Within the field of mining software repositories, many approaches, such as topic modeling and concept location, rely on the automated application of machine learning algorithms to corpora. Unfortunately the output of these tools is often difficult to distinguish and interpret as they are often so abstract. Thus to have a meaningful discussion about the topics of software development, we must be able to devise appropriate labels for extracted topics. However, these approaches neither use domain-specific knowledge to improve results, nor contextualize those results for developers. While too much specificity can produce non-generalizable results, too little produces broad learners that do not provide much immediately useful detail. This paper implements *labelled topic extraction*, in which topics are extracted from commit comments and given labels relating to a cross-project taxonomy. We focus on non-functional requirements related to software quality as a potential generalization, since there is some shared belief that these qualities apply broadly across many software systems and their development artifacts. We evaluated our approach with an experimental study on two large-scale database projects, MySQL and MaxDB. We extracted topics using Latent Dirichlet Allocation (LDA) from the commit log comments of their version control systems (CVS and BitKeeper). Our results were generalizable across the two projects, showing that non-functional requirements were commonly discussed, and we identified topic trends over time. Our labelled topic extraction technique allowed us to devise appropriate, context-sensitive labels across these two projects, providing insight into software development activities.

1. INTRODUCTION

What's in a name? that which we call a rose
By any other name would smell as sweet;

– Romeo and Juliet, II:ii

Few would argue that software development is a bed of roses. To most of us, the important properties of a rose are sensual in nature, and concern appearance, odour, and feel. While horticulturalists and botanists may see more semantic depth than this, one rose is often just as pleasing to us as another. Software development artifacts, on the other hand, are abstract and intangible. They are often unnamed and ephemeral — even more so than roses — and maybe partially derived from other unnamed and ephemeral artifacts. Yet to have meaningful discussions about how software development is progressing, we must be able to devise appropriate labels for our development topics and be able to categorize the development artifacts as belonging to one or more of these topics.

Topics arise from the wide variety of issues which occur during a software project's life-cycle. These topics can relate to, among others, the problem domain, the processes and tools used, or the development artifacts themselves. The set of development topics for a given project can sometimes be extracted automatically by analyzing artifacts within software repositories, such as change-log comments that developers create when committing revisions to the project's source control system. Our previous work dealt with topic trends, which are topics that recur over time [10]. We observed that topic trends were often non-functional requirements.

Topics in this paper and the previous paper are word bags or word distributions. These word distributions are found via Latent Dirichlet Allocation [2], which finds independent word distributions shared among documents (change-log comments). The unfortunate aspect of topics that are word distributions is that they lack the tangibility of a rose. They do not self identify with their tangible properties. Topics have to be identified by interpreting the prevalent words

in the word distribution and by inspecting related documents. This is impractical when one has to handle more than one hundred different topics. It would be nice if we could have automatic assistance to determine what the topic is about. This is the power of labelling and naming.

A topic — that is, a word distribution in our world — needs a suggestive name to succinctly convey its intent, and make it easy to use in discussions and analyses about the development of the system. Our previous experience leads us to believe that automated topic naming is currently infeasible in the general case. We have therefore decided to focus on topic labels from the sub-domain of non-functional requirements related to software quality.

Traditionally topic extraction has required manual annotation to derive domain-relevant labels. This paper implements *labelled topic extraction*, in which topics are extracted and given labels relating to a cross-project taxonomy. This is an ontological approach where we attempt to relate networks of words to labels and then search for these terms within our topics. We also compare this approach to machine learners.

Our contributions include:

- We introduce labelled topic extraction, and demonstrate its usefulness compared to other approaches.
- We show that these labels with their topics can be learnt and used to classify other data-sets.
- We present visualizations of named topics and their trends over time to aid communication and analysis.
- We use an exploratory case study of several open source database systems to show how named topics can be compared between projects.

We first introduce some important terminology for our work. We then describe our methodology, including our data-sets, then highlight our results. We conclude with a look at related work and possible improvements.

2. BACKGROUND

We provide a brief overview of software repository mining and information retrieval. This work is related to the mining software repositories (MSR) [12] research community as it deals expressly with analyzing a project's source control repository, and the messages associated with revisions therein.

2.1 Definitions

We will use the following terminology 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 a developer 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 ten 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. A *label* is part of a title we attach to a topic, whether manually or automatically.

Area of ROC Curve is the area under the Receiver Operating Characteristic (ROC) curve, sometimes referred to as AUC. ROC values reflect a score, similar to school lettergrades (A is 0.9, C is 0.6), reflecting how well a particular learner performed for the given data. A ROC result of 0.5 would be equivalent to a random learner (randomly classifying data). ROC maps to the more familiar concepts of precision/sensitivity and recall/specificity: it plots the true positive rate (sensitivity) versus the false positive rate (1 - specificity). A perfect learner has a ROC value of 1.0, reflecting perfect recall and precision.

2.2 Topic and Concept Extraction

Topic extraction, sometimes called concept extraction, uses tools such as Latent Dirichlet Allocation (LDA) [2] and Latent Semantic Indexing (LSI) to extract independent word distributions (topics) from documents (commit log comments). Many researchers [17, 22, 16, 15] have applied tools like LSI and LDA to mining software repositories, in particular analyzing source code, bugs or developer comments.

Typically a topic analysis tool like LDA will try to find N independent word distributions found within the word distributions of all the messages. Linear combinations of these N word distributions are then meant to be able to recreate the word distributions of all of the underlying messages. These N word distributions effectively form topics: cross cutting collections of words relevant to one or more documents. Our problem is that these topics are not easy to interpret, as the underlying pattern is not clear. We feel that automatic labelling or naming of these topics would be helpful with respect to interpreting the subject of a topic. LDA extracts topics in an unsupervised manner; the algorithm relies solely on the source data with no human intervention.

In topic analysis a single document, such as a commit message, can be related to multiple topics. Representing documents as a mixture of topics maps well to source code repository commits, which often have more than one purpose [10]. 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 paper a topic is a set of tokens extracted from commit messages found within a project's source control system (SCS).

One issue that arises with use of unsupervised techniques is how to label the topics. While the topic models themselves are generated automatically, what to make of them is less clear. For example, in our previous work [10], as well as in Baldi et al. [1], topics are named manually: human experts read the highest-frequency members of a topic and assign a

keyword accordingly. E.g., for the word list *"listener change remove add fire"*, Baldi et al. assign the keyword *event-handling*. The labels are reasonable enough, but still require an expert in the field to determine them. Our technique is automated, because we match keywords from WordNet [8] to words in the topic model.

2.3 Supervised learning

While unsupervised techniques (LSI and LDA are both unsupervised) are appealing in their lack of human intervention, and thus lower effort, supervised learners have the advantage of domain knowledge which typically means improved results. In supervised learning, the data is divided into slices. One slice is manually annotated by the domain expert, and the classifications he/she determines are applied to the remaining slices. In this paper, we employ the WEKA [9] and Mulan [24] machine learning frameworks in order to test supervised learning.

3. METHODOLOGY

To evaluate our approach, we sought candidate systems that were mature projects and had openly accessible source control repositories. We also decided to select systems from the same application domain, as we felt the functional requirements would probably be broadly similar. We selected MySQL and MaxDB as they were open-source, partially-commercial database systems. MaxDB started in the late 1970s as a research project, and was later acquired by SAP. As of version 7.500, released April 2007, the project has 940 thousand lines of C source code [1]. The MySQL project started in 1994 but MySQL 3.23 was released in early 2001. MySQL contains 320 thousand lines of C and C++ source code.

3.1 Generating the data

For each project, we used source control commit comments, the messages that programmers write when they commit revisions to a source control repository. We leveraged the data that we gathered in [10] for this work. An example of a typical commit message is: *"history annotate diffs bug fixed (if mysql_real_connect() failed there were two pointers to malloc'ed strings, with memory corruption on free(), of course)"*. We extracted these messages and indexed them by creation time. We summarized each message as a word distribution but removed stop-words such as common English words like *the* and *at*.

From that data-set, we created an XML file which separated commits into monthly windows. This size of period is smaller than the time between minor releases but large enough for there to be sufficient commits to analyze. We applied Blei's LDA implementation [2] against the word distributions of these commits, and generated lists of topics per period. We arbitrarily set the number of topics to generate to 20, 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. As well, more than 20 topics quickly became infeasible for inspection and it was difficult to discern the difference in topics.

[1]generated using David A. Wheeler's *SLOCCount*.

3.2 Associating labels

Topics are word distributions: essentially lists of words ranked by frequency, which can be burdensome to interpret and hard to distinguish and understand. Once we had topics for each period, we tried to associate them with a label from a list of keywords and related terms. We performed simple string matching between these topics and our lists, 'naming' a topic if it contained that word or words. We used several different word lists for comparison.

Our first word list set, exp1, was generated using the ontology described in Kayed et al. [13]. That paper constructs an ontology for software quality measurement using eighty source documents, including research papers and international standards. The labels we used:

> integrity, security, interoperability, testability, maintainability, traceability, accuracy, modifiability, understandability, availability, modularity, usability, correctness, performance, verifiability, efficiency, portability, flexibility, reliability.

Our second word list set, exp2, relied on the ISO quality model (ISO9126) [11]. ISO9126 describes six high-level quality requirements (listed in Table 1). ISO9126 is "an international standard and thus provides an internationally accepted terminology for software quality [3, p. 58]," that is sufficient for the purposes of this research. However, the terms extracted from ISO9126 may not capture all words associated with the labels. For example, the term "redundancy" is one most would agree is relevant to discussion of reliability, but is not in the standard. We therefore took the words from the taxonomy and expanded them.

To construct these expanded word-lists, we used WordNet [8], an English-language "lexical database" that contains semantic relations between words, including meronymy and synonymy. We then added Boehms 1976 software quality model [4], and classified his eleven ilities into their respective ISO9126 qualities. We did the same for the quality model produced by McCall et al. [18]. Finally, we analyzed two mailing lists from the KDE project to enhance the specificity of the sets. We selected KDE-Usability, which focuses on usability discussions for KDE as a whole; and KDE-Konqueror, a mailing list about a long-lived web browser project. For each high-level quality in ISO9126, we first searched for our existing labels; we then randomly sampled twenty-five mail messages that were relevant to that quality, and selected co-occurring terms relevant to that quality. For example, we add the term "performance" to the synonyms for *efficiency*, since this term occurs in most KDE mail messages that discuss efficiency.

For the third – exp3 – list of quality labels, we extended the list from exp2 using unfiltered WordNet similarity matches. Similarity in WordNet means siblings in a hypernym tree. We do not include these words here for space considerations (but see the Appendix for our data repository). It is not clear the words associated with our labels are specific enough, however: for example, the label *maintainability* is associated with words *ease* and *ownership*.

3.3 Supervised learning

Label	Related terms
Maintainability	testability changeability analyzability stability maintain maintainable modularity modifiability understandability + interdependent dependency encapsulation decentralized modular
Functionality	security compliance accuracy interoperability suitability functional practicality functionality + compliant exploit certificate secured buffer overflow policy malicious trustworthy vulnerable vulnerability accurate secure vulnerability correctness accuracy
Portability	conformance adaptability replaceability installability portable movableness movability portability + specification migration standardized l10n localization i18n internationalization documentation interoperability transferability
Efficiency	resource behaviour time behaviour efficient efficiency + performance profiled optimize sluggish factor penalty slower faster slow fast optimization
Usability	operability understandability learnability useable usable serviceable usefulness utility useableness usableness serviceableness serviceability usability + gui accessibility menu configure convention standard feature focus ui mouse icons ugly dialog guidelines click default human convention friendly user screen interface flexibility
Reliability	fault tolerance recoverability maturity reliable dependable responsibleness responsibility reliableness reliability dependableness dependability + resilience integrity stability stable crash bug fails redundancy error failure

Table 1: Qualities and associated signifiers WordNet version (exp2)

In order to validate how effective these word-bag approaches to topic labelling would be we had to make a data set to test against. For MySQL 3.23 and MaxDB 7.500, we manually annotated each extracted topic in each period with the same quality labels as `exp2` (software qualities). We looked at each period's topics, and assessed what the data – consisting of the frequency-weighted word lists and messages – suggested was the topic for that period. We were able to pinpoint the appropriate label using auxiliary information as well, such as the actual revisions and files that were related to this topic. For example, for the MaxDB topic consisting of a message "exit() only used in non NPTL LINUX Versions", we tagged that topic *portability*. We compared against this data-set, but we also used it for our supervised learning based topic classification.

We first compared our previous analysis using label matching to our manual classifications to get an error rate for that process described below in Section 4.2.

For supervised learning, we used a suite of supervised classifiers from WEKA [9]. WEKA contains a suite of machine learning tools such as support vector machines and Bayes nets. We also used the multi-labelling add-on for WEKA, Mulan [24][2]. Traditional classifiers map our topics to a single class, whereas Mulan allows for a mixture of classes per topic, which maps to what we observed while manually labelling topics.

To assess the performance of the supervised learners, we did a 10-fold cross-validation. This is when a set is partitioned into 10 partitions and then each partition is used once as a test set and 9 other times as part of the training set of 9 partitions. We have reported these results below in Section 4.3.

[2]`http://mlkd.csd.auth.gr/multilabel.html`

Finally, using this data, we evaluated two research questions (see Section 4.7):

1. Do label frequencies change over time? That is, is a certain quality of more interest at one point in the life-cycle than some other?

2. Do the different projects differ in their relative topic interest? That is, is a particular quality more important to one project than the other projects?

4. OBSERVATIONS AND EVALUATION

4.1 Word list similarity

In general, this approach did not work out well as common labels dominated the less common labels. The related words for *correctness*, for example, tended to be too lengthy and non-specific. Table 2 lists results. A *named topic* is a topic with a matching label. There are {periods X 20} topics per project as we told LDA to extract 20 topics per period. All experiments were run on MaxDB 7.500 and MySQL 3.23 data.

For `exp1`, our best performing labels (the labels matched with the most topics) were *correctness* (182 topics) and *testability* (121). We did not get good results for usability or accuracy, which were associated with fewer than ten topics. We also looked for correlation between our labels: Excluding double matches (self-correlation), our highest co-occurring

Measure	exp1	exp2	exp3
Topics	500	500	500
Named topics	281	125	328
Unnamed topics	139	295	92

Table 2: Automatic topic labelling for MaxDB 7.500

terms were verifiability and traceability, and testability and correctness (76 and 62 matches, respectively).

For exp2, there are many more unnamed topics. Only reliability produces a lot of matches, mostly with the word 'error'. Co-occurrence results were poor.

For exp3, we had many more named topics. As we mentioned, the word-lists are quite broad, so there are likely to be false-positives. See the following sections for our error analysis. We found a high of 265 topics for usability, with a low of 44 topics for maintainability. Common co-occurrences were reliability and usability, efficiency and reliability, and efficiency and usability (200, 190, and 150 topics in common, respectively).

4.2 Analysis of the unsupervised labelling

Based on the labels, and our manual topic labelling, we compared the results of the unsupervised word matching approach. For each quality we tried to assess whether the manual tag matched the unsupervised label assigned. Table 3 shows our results for MaxDB and MySQL. In general results are poor. Using the F-Measure, the weighted average of precision and recall, where 1 is perfect, our best results are 0.6, a few at 0.4, and most around 0.2. We achieved similar results using the Matthew's correlation coefficient (used to measure efficacy where classes are of different sizes) and ROC.

Based on these results we find that reliability and usability worked well for MaxDB in exp2 and better in exp3. MySQL had reasonable results within exp2 for reliability and efficiency. MySQL's results for efficiency did not improve in exp3 but other qualities such as functionality did improve. If a C grade performance has a ROC value of 0.6 then most of these tests scored a grade of C or less, but our results were still better than random chance.

4.3 Analysis of the supervised labelling

We took our annotated data-set and applied supervised learners to it. Because our data-set was of word counts we expected Bayesian techniques, often used in spam filtering, to perform well. We also tried other learners that WEKA [9] provides: rule learners, tree learners, vector space learners, and support vector machines. Table 4 shows the performance of the best performing learner per label. We considered the best learner for a label to be the one which had the highest ROC value for that label. Table 4 uses the ZeroR learner as a baseline, since it naively chooses the largest category all of the time. The ZeroR difference is often negative. For labels which are not as common, this can be expected because any miscategorization will hurt accuracy. This is why the F1 (F-measure) and the ROC values are useful, as they can better present performance on labels which are not applicable to the majority of samples.

Table 4 shows that MaxDB and MySQL have quite different results, as the ROC values for reliability and functionality seem swapped between projects. It should be noted that for both projects Bayesian techniques did the best out of a wide variety of machine learners tested. Discriminative Multinomial Naive Bayes (DMNBtext), Naive Bayes (NaiveBayes) and Multinomial Naive Bayes (NaiveBayesMultinomal) are all based on Bayes's theorem and all assume, naively, that the features are independent. The features we used are word counts per message. One beneficial aspect of this result is that it suggests we can have very fast training and classifying since Naive Bayes can be calculated in $O(N)$ for N features.

The smaller the label the harder it is to get accurate results. Nevertheless, these results are better than our previous word bag results of exp2 and exp3, because the ROC values are sufficiently higher in most cases (other than MaxDB reliability and MySQL efficiency).

The limitation of the approach we took here is that we assume labels are independent; however, labels could be correlated with each other. We also did not evaluate how well the learners performed together.

4.4 Applying multiple labels to topics

We applied the Mulan [24] library for Multi-Label learning to our data-set because intuitively, topics can have more than one label, much like how a particular source code revision can have more than one topic. Multi-label learning is more than just classifying entities with more than one label. It also includes methods for determining the performance of such techniques. The problem framed in the learners above has changed; instead of looking at the precision and recall of applying one label, we rank multiple labels at once. We must check if the full subset of labels was applied, and then how much of that subset was applied.

Another aspect of multi-label learning are micro versus macro measurements. Macro measurements are aggregated at a class or label level. Micro measurements are aggregated at a decision level. So a macro-ROC measurement is the average ROC over the ROC values for all labels, where a micro-ROC is the average ROC over all examples that were classified. Unfortunately for MaxDB, the macro-ROC values are undefined because of poor performance of one of the labels.

We have presented the results of Mulan's multi-label learners in Table 5. Binary Relevance (BR), Calibrated Label Ranking (CLR) and Hierarchy Of Multi-label classifiERs (HOMER), performed the best. HOMER and BR act as a hierarchy of learners: BR is flat, while HOMER tries to build a deeper hierarchy to build a more accurate learner [24]. These classifiers performed better than other multi-label classifiers. They have the best micro and macro ROC scores, although their results seem comparable to the naive Bayesian learners we used in Section 4.3.

4.5 Summary of techniques

Very rarely did exp2 and exp3 (naive word matching) ever perform as well as the machine learners. For MaxDB, relia-

Experiment	Label	F1	MCC	Precision	Recall	ROC
MaxDB exp2	portability	0.228	0.182	0.520	0.146	0.553
	efficiency	0.217	0.125	0.237	0.200	0.558
	reliability	0.380	0.340	0.246	0.829	0.765
	functionality	0.095	0.083	0.250	0.059	0.521
	maintainability	0.092	0.123	0.571	0.050	0.520
	usability	0.175	0.138	0.113	0.389	0.620
	total	0.236	0.127	0.248	0.225	0.561
MySQL exp2	portability	0.138	0.211	1.000	0.074	0.537
	efficiency	0.345	0.327	0.476	0.270	0.625
	reliability	0.425	0.287	0.348	0.545	0.669
	functionality	0.025	0.006	0.571	0.013	0.501
	maintainability	0.000	0.000	0.000	0.000	0.500
	usability	0.175	0.135	0.200	0.156	0.560
	total	0.167	0.095	0.403	0.105	0.527
MaxDB exp3	portability	0.472	0.286	0.402	0.573	0.660
	efficiency	0.223	0.068	0.130	0.778	0.549
	reliability	0.257	0.196	0.149	0.927	0.652
	functionality	0.236	0.187	0.138	0.824	0.665
	maintainability	0.338	0.112	0.266	0.463	0.566
	usability	0.108	0.094	0.057	0.944	0.595
	total	0.258	0.093	0.160	0.671	0.568
MySQL exp3	portability	0.413	0.170	0.564	0.325	0.574
	efficiency	0.158	0.105	0.089	0.703	0.608
	reliability	0.388	0.240	0.260	0.758	0.660
	functionality	0.652	0.240	0.655	0.649	0.620
	maintainability	0.203	0.007	0.240	0.175	0.503
	usability	0.105	0.013	0.057	0.688	0.513
	total	0.362	0.076	0.284	0.499	0.544

Table 3: Results for automatic topic labelling. F1 = f-measure, MCC = Matthew's correlation coeff., ROC = Area under ROC curve

Label	Project	Learner	ROC	ZeroR Acc.	ZeroR. Diff	F1
portability	MySQL	NaiveBayesMultinomial	0.74	58.53	11.43	0.62
efficiency	MySQL	NaiveBayes	0.67	93.69	-7.68	0.23
reliability	MySQL	NaiveBayes	0.73	83.11	-12.80	0.41
functionality	MySQL	DMNBtext	0.81	54.44	21.67	0.77
maintainability	MySQL	DMNBtext	0.78	76.62	3.41	0.32
usability	MySQL	NaiveBayes	0.75	94.54	-5.80	0.21
portability	MaxDB	NaiveBayes	0.84	77.12	2.06	0.61
efficiency	MaxDB	NaiveBayes	0.62	88.43	-11.31	0.25
reliability	MaxDB	DMNBtext	0.84	89.46	3.86	0.57
functionality	MaxDB	NaiveBayes	0.67	91.26	-6.94	0.31
maintainability	MaxDB	NaiveBayes	0.70	79.43	-9.25	0.42
usability	MaxDB	NaiveBayes	0.56	95.37	-4.37	0.00

Table 4: Per label, per project, the best learner for that label. ROC value rates learner performance, compared with the ZeroR learner (a learner which just chooses the largest category all of the time). F1 is the F-measure for that particular learner.

Performance Metric	MySQL BR	MySQL CLR	MySQL HOMER	MaxDB BR	MaxDB CLR	MaxDB HOMER
example Subset Accuracy	0.19	0.24	0.31	0.40	0.43	0.45
label macro-ROC	0.74	0.66	0.64	NaN	NaN	NaN
label macro-F1	0.46	0.30	0.43	0.32	0.19	0.23
label macro-Precision	0.41	0.41	0.49	0.29	0.31	0.32
label macro-Recall	0.56	0.26	0.40	0.38	0.15	0.20
label micro-ROC	0.81	0.81	0.77	0.76	0.66	0.62
label micro-F1	0.59	0.53	0.61	0.39	0.27	0.34
label micro-Precision	0.51	0.69	0.66	0.34	0.42	0.49
label micro-Recall	0.68	0.43	0.56	0.47	0.21	0.26
rank Avg. Precision	0.76	0.77	0.69	0.54	0.57	0.54
rank Coverage	1.51	1.47	1.94	1.40	1.23	1.43
rank One-error	0.40	0.39	0.47	0.81	0.79	0.81
rank Ranking Loss	0.19	0.18	0.27	0.41	0.35	0.41

Table 5: MySQL and MaxDB Mulan results per Multi-Label learner

bility was slightly better detected using the static word list of exp2. In general, the machine learners and exp3 did better than exp2 for both MaxDB and MySQL. For both MySQL and MaxDB usability was better served by exp2. Usability was a very infrequent label, however.

We found that the multi-label learners of BR, CLR and HOMER did not do as well for Macro-ROC and Micro-F1 as NaiveBayes and other NaiveBayes derived learners did. This suggests that by sticking together multiple NaiveBayes learners we could probably label sets of topics effectively, but it would require a separate NaiveBayes learner per label.

4.6 Visualization

We have created two visualizations of the manually labelled data. A problem we faced while visualizing was how to display tag overlaps. Our solution, while not optimal, was to assign a separate colour to each distinct set of annotations. Figures 1 and 2 show extracted topics grouped by annotations. Annotations or labels that occur in adjacent windows are joined together as trends. For MaxDB (Figure 1) the largest trends were related to maintainability and portability. MaxDB supports numerous platforms and portability was a constant issue facing the project's development. MySQL was different, with many topics overlapping, so there were many different subsets. Functionality trends were prevalent throughout its history (the largest reddish streak across the top). Two combination tags of "functionality portability" (orange) and "maintainability portability" (teal) streaked across most of the history of MySQL. Since this was MySQL 3.23, a stable branch, we expect that issues dealing with portability and maintenance would be the primary concerns of this branch: developers probably wanted it to work with current systems and thus had to update it.

4.7 Comparing MaxDB and MySQL

We observed that MySQL had more topic subsets than MaxDB. MySQL 3.23 also had more topics and a longer recorded history than MaxDB 7.500. We tagged more MySQL topics with annotations than MaxDB topics yet both shared similarities. In terms of non-functional requirements both projects had long running trends that focused on functionality, maintainability, and portability, yet MaxDB had more of a focus on efficiency and reliability. MaxDB differed from MySQL since MaxDB was being actively developed, whereas halfway through our history of MySQL 3.23, other versions of MySQL were being actively developed: MySQL versions 4.0, 4.1, and eventually 5.0 and 5.1. In other words, MySQL 3.23 was being maintained rather than actively developed, whereas MaxDB 7.500 was being actively developed into MaxDB 7.6, and then maintained thereafter.

MySQL and MaxDB's machine learners did make decisions based off some shared words. Words that were used to classify topics that were shared between MySQL and MaxDB included: bug, code, compiler, database, HP UX, delete, memory, missing, problems, removed, add, added, changed, problem, and test. Adding these words to the word bags of exp2 and exp3 could improve performance while ensuring they were only domain specific.

With respect to the questions raised in section 3.3:

Do label frequencies change over time? – Yes, MySQL's label frequencies decreased as it got older. Usability and reliability labels became more and more infrequent as it matured. Maintainability topics became more prevalent as MaxDB matured.

Do the different projects differ in their relative topic interest? – Yes. MySQL 3.23 had proportionally more functionality labelled topics, while MaxDB had proportionally more efficiency and portability related topics.

4.8 Annotation observations

We found many topics that were not non-functional requirements (NFRs) but were often related to them. For instance, concurrency was mentioned often in the commit logs and was related to correctness and reliability, because concurrency was troublesome. Configuration management and source control related changes appeared often and sometimes there were topics dedicated to configuration management. These kinds of changes are slightly related to maintainability. A non-functional change that was not quality-related was licensing and copyright; many changes were simply to do with updating copyrights or ensuring copyright or license headers

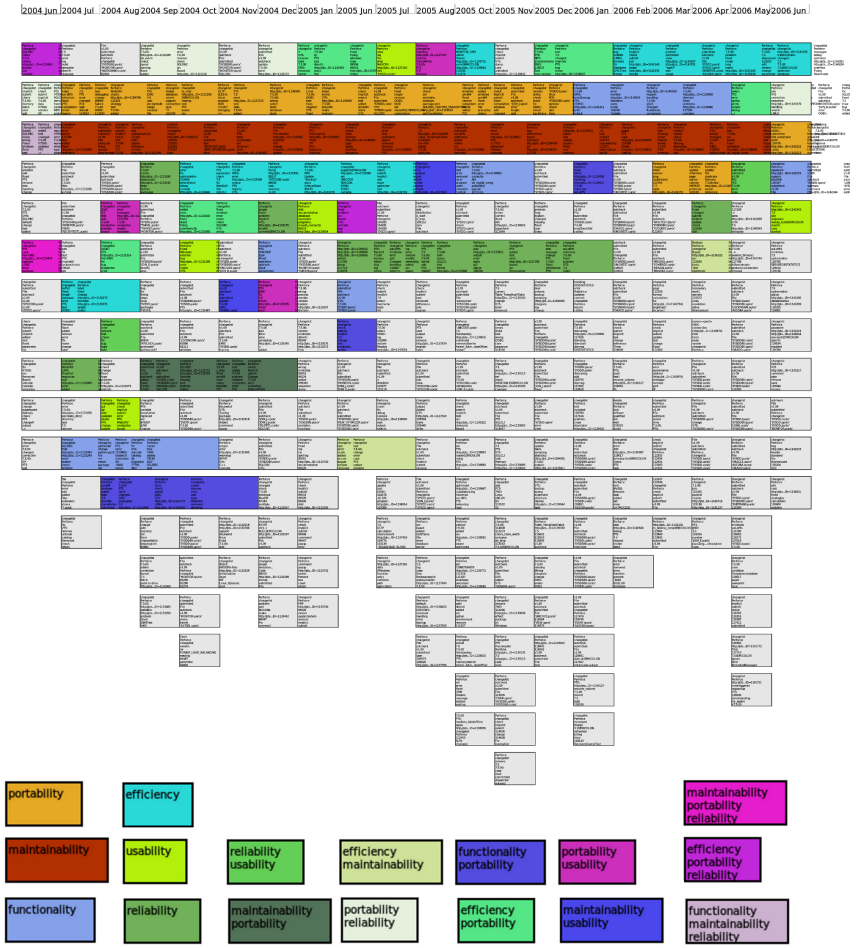

Figure 1: MaxDB 7.500: Labelled topics related to non-functional software qualities plotted over time, continuous topics are trends that occur across time

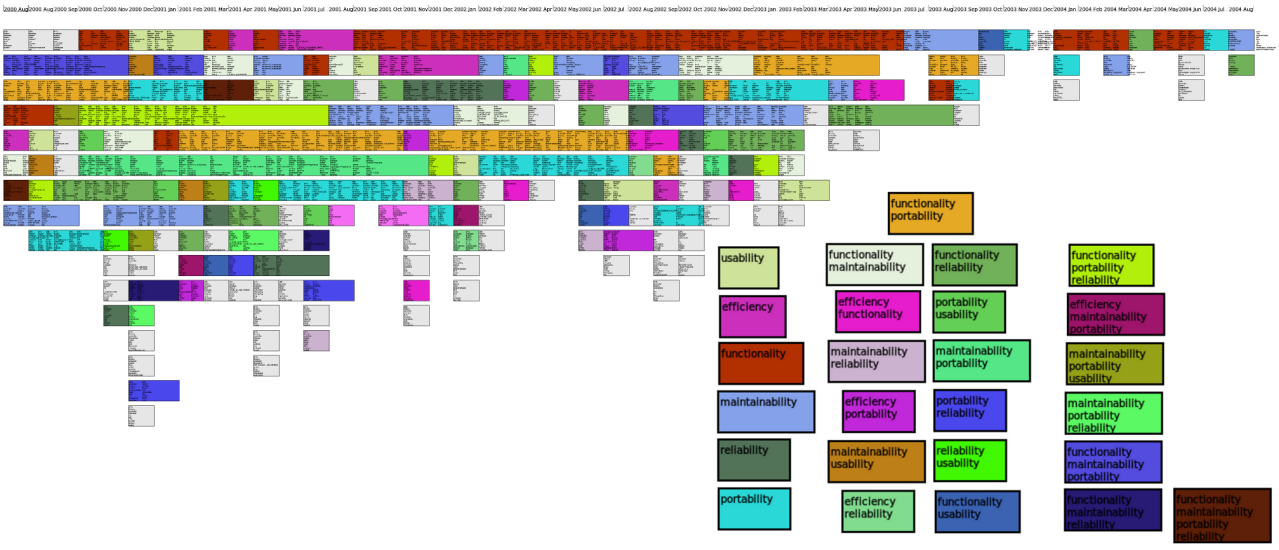

Figure 2: MySQL 3.23: Labelled topics related to non-functional software qualities plotted over time, continuous topics are trends that occur across time

were applied to files.

We noticed that occasionally the names of modules would conflict with words related to other non-functional requirements. For instance, optimizers are very common modules in database systems: both MySQL and MaxDB have optimizer modules. In MySQL the optimizer is mentioned but often the change deals with correctness or another quality. Despite this difference, the name of the module could fool our learners into believing the change was always about efficiency. Perhaps a project specific word-bag is needed in order to avoid automated mistakes due to the names of entities and modules of a software project.

4.9 Effectiveness

With ROC values ranging from 0.6 to 0.8 we can see there is promise in these attempts. exp2 and exp3 both indicate that static information can be used to help label topics without any training whatsoever. If the techniques used in exp2 and exp3 were combined with the supervised techniques we could probably reduce the training effort. Both Naive Bayesian learners and the word-list approaches were computationally efficient. These results are promising, because the results are accurate enough to be useful, while still cheap enough to execute to be feasible as an automated or semi-automated method of labelling topics by their software qualities.

4.10 Threats to validity

Our study suffers from multiple threats to validity. *Construct validity* issues include that we used only commit messages rather than mail or bug tracker messages. We also chose our taxonomy and the data to study. *Internal validity* issues are to do with inter-rater reliability. *External validity* issues are that our data originated from OSS database projects and thus might not be applicable to commercially developed software. Furthermore, our analysis techniques rely on a project's use of meaningful commit messages.

5. RELATED WORK

The idea of extracting higher-level 'concerns' (also known as 'concepts', 'aspects' or 'requirements') has been approached in two ways.

Cleland-Huang and her colleagues published work on mining requirements documents for non-functional requirements (NFR) (quality requirements) [6]. One approach they tried was similar to this one, with keywords mined from NFR catalogues found in their previous work [5]. They managed recall of 80% with precision of 57% for the Security NFR, but could not find a reliable source of keywords for other NFRs. Instead, they developed a supervised classifier by using human experts to identify an NFR training set. There are several reasons we did not follow this route. One, we believe we have a more comprehensive set of terms due to the taxonomy we chose. Secondly, we wanted to compare across projects. Their technique was not compared across different projects and the applicability of the training set to different corpora is unclear. A common taxonomy allows us to make inter-project comparison (subject to the assumption that all projects conceive of these terms in the same way). Thirdly, while the objective of Cleland-Huang's study was to identify new NFRs (for system development) our study assumes these NFRs are latent in the textual documents of the project. Finally, the source text we use is less structured than their requirements documents.

In the same vein, Mockus and Votta [21] studied a large-scale industrial change-tracking system. They also leveraged WordNet, but only for word roots. They felt the synonyms would be non-specific and cause errors. A nice contribution was access to system developers, with whom they could validate their labels. Since we try to bridge different organizations, these interviews are infeasible (particularly in the distributed world of open-source software).

The other approach is to start with code repositories, and try to extract concerns from there. Marcus et al. [17] describe their use of Latent Semantic Indexing to identify commonly occurring concerns for software maintenance. Some results were interesting, but their precision was quite low. ConcernLines [23] shows tag occurrence using colour intensity. They mined change request tags (such as 'milestone 3') and used these to make evolutionary analyses of a single product. The presence of a well-maintained set of tags is obviously essential to the success of this technique.

Mens et al. [20] conducted an empirical study of Eclipse, the open source software (OSS) source code editor, to verify the claims of Lehman [14]. They concerned themselves with source code only, and found Law Seven, "Declining Quality", to be too difficult to assess: "[we lacked an] appropriate measurement of the evolving quality of the system as perceived by the users [20, p. 388]". This paper examines the notions of quality in terms of a consistent ontology, as Mens et al. call for in their conclusions.

Mei et al. [19] use context information to automatically name topics. They describe probabilistic labelling, using the frequency distribution of words in a topic to create a meaningful phrase. They do not use external domain-specific information as we do. In [7], we describe our earlier project, similar to this, to identify change in quality requirements in GNOME software projects; our approach is solely text-matching, however, it does not leverage machine learning strategies.

6. CONCLUSIONS AND FUTURE WORK

We demonstrated that static but domain-specific knowledge can improve unsupervised labelling of extracted topics. Our exp2 experiment used small accurate word bags to label topics but performed just as well as exp3, which used many more general terms from WordNet. We then showed that with some supervision, and by using efficient machine learners based on Naive Bayesian classifiers, we could improve the accuracy of automatic labelling topics even further.

Our manual inspection and annotation of the topics extracted from MySQL and MaxDB revealed that many of the extracted topics dealt with non-functional requirements, and these topics were spread across the entire history of a project. In the cases of MaxDB and MySQL, portability was a constant maintenance concern and was prevalent throughout the entire lifetime of the projects.

We showed that non-functional requirements are often trend-

ing topics, that non-functional requirements are quite common in developer topics, and that there are efficient methods of semi-automating and automating topic labelling.

There are many avenues of further investigation. We want to investigate developer attitudes related to these labels: i.e., when we label a topic, was the developer expressing positive or negative qualities about that label? It is difficult to map abstract qualities to particular messages. Is a "quality" discussion about more than just corrective maintenance?

7. APPENDIX

Our data and scripts are available at `http://softwareprocess.es/nomen/`

8. REFERENCES

[1] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya. A theory of aspects as latent topics. In *Conference on Object Oriented Programming Systems Languages and Applications*, pages 543–562, Nashville, 2008.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3(4-5):993–1022, May 2003.

[3] J. Bøegh. A New Standard for Quality Requirements. *IEEE Software*, 25(2):57–63, 2008.

[4] B. Boehm, J. R. Brown, and M. Lipow. Quantitative Evaluation of Software Quality. In *International Conference on Software Engineering*, pages 592–605, 1976.

[5] L. Chung, B. A. Nixon, E. S. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*, volume 5 of *International Series in Software Engineering*. Kluwer Academic Publishers, Boston, October 1999.

[6] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. In *International Requirements Engineering Conference*, pages 39–48, Minneapolis, Minnesota, 2006.

[7] N. A. Ernst and J. Mylopoulos. On the perception of software quality requirements during the project lifecycle. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, Essen, Germany, June 2010. in press.

[8] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[10] A. Hindle, M. W. Godfrey, and R. C. Holt. What's hot and what's not: Windowed developer topic analysis. In *International Conference on Software Maintenance*, pages 339–348, Edmonton, Alberta, Canada, September 2009.

[11] Software engineering – Product quality – Part 1: Quality model. Technical report, International Standards Organization - JTC 1/SC 7, 2001.

[12] H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.*, 19(2):77–131, 2007.

[13] A. Kayed, N. Hirzalla, A. Samhan, and M. Alfayoumi. Towards an ontology for software product quality attributes. In *International Conference on Internet and Web Applications and Services*, pages 200–204, May 2009.

[14] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski. Metrics and laws of software evolution-the nineties view. In *International Software Metrics Symposium*, pages 20–32, Albuquerque, NM, 1997.

[15] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining Eclipse Developer Contributions via Author-Topic Models. *International Workshop on Mining Software Repositories at ICSE*, pages 30–30, May 2007.

[16] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. Source Code Retrieval for Bug Localization Using Latent Dirichlet Allocation. In *Working Conference on Reverse Engineering*, pages 155–164, Antwerp, Belgium, 2008.

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

[18] J. McCall. *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager*, volume 1-3. General Electric, November 1977.

[19] Q. Mei, X. Shen, and C. Zhai. Automatic labeling of multinomial topic models. In *International conference on Knowledge discovery and data mining*, pages 490–499, San Jose, California, 2007.

[20] T. Mens, J. Fernandez-Ramil, and S. Degrandsart. The evolution of Eclipse. In *International Conference on Software Maintenance*, pages 386–395, Shanghai, China, October 2008.

[21] A. Mockus and L. Votta. Identifying reasons for software changes using historic databases. In *International Conference on Software Maintenance*, pages 120–130, San Jose, CA, 2000.

[22] D. Poshyvanyk and A. Marcus. Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code. In *International Conference on Program Comprehension*, pages 37–48, June 2007.

[23] C. Treude and M.-A. Storey. ConcernLines: A timeline view of co-occurring concerns. In *International Conference on Software Engineering*, pages 575–578, Vancouver, May 2009.

[24] G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining multi-label data. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*. Spring, 2nd edition, 2010.