Automated topic naming to support analysis 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

Researchers have used topic modeling and concept location to understand the latent topics of software development artifacts. These techniques use unsupervised machine-learning algorithms to recover topics. These topics are word-lists and are difficult to distinguish and interpret. Topics are not meaningful until they have been named or interpreted. Current topic labelling approaches are manual, and do not use domain-specific knowledge to improve, contextualize, or describe results for the developers. We propose a solution: *labelled topic extraction*. Topics are extracted using Latent Dirichlet Allocation (LDA) from commit-log comments recovered from source control systems such as CVS and BitKeeper. These topics are given labels relating to a generalizable cross-project taxonomy consisting of non-functional requirements. Our approach was evaluated with experiments and case studies on two large-scale RDBMS projects: MySQl and MaxDB. Labelled topic extraction produces appropriate, context-sensitive labels relevant to these projects, which provides fresh insight into their evolving software development activities.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Managment—*Lifecycle*; D.2.1 [**Software Engineering**]: Requirements/Specifications—*Tools*

Keywords

Topic analysis, LDA, non-functional requirements

1. INTRODUCTION

Topics in software engineering practice refer to specific shared threads of discussions between project stakeholders. These topics can include particular requirements, maintenance requests, bug fixes or other software engineering tasks. While addressing a topic of development, stakeholders often produce or modify multiple development artifacts. These artifacts can include source code, bug reports, revisions to source code, and test cases. Topics are typically latent, i.e., not explicitly named, and can be in several potential states: they may not be immediately resolved, they may re-occur, yet others are only dealt with briefly (e.g., a bug report that is closed).

Developers often interact with abstractions of the artifacts from which topics are derived, rather than the artifacts themselves. Such abstractions include project dashboards [16], tags [29], and complexity metrics [22]. Without the structured information which dashboards typically rely on, topics become a valuable and automatically extractable abstraction of software artifacts. Topics abstract software artifacts by their subject matter, rather than their concrete representation. Much like aspect-oriented programming, topic labeling can identify cross-cutting concerns in a software project (as in [1]). The topics of development are relevant because they identify the distinct issues facing stakeholders at a particular time.

The goal of concept location and topic analysis research is to recover these topics and the related software artifacts. To date, the results of topic analysis and concept location are project specific, and not generalizable. Much of the current research stops short of automatically interpreting the purpose of the topic, e.g. [1, 18, 20]. That is, extracted topics are usually manually named and labelled by those who read the topic words.

Deriving and labelling topics is a challenging machine learning problem. Topic analysis produces unlabelled context-free word-lists or word distributions. Topics have to be identified by interpreting the prevalent words in a word distribution and by inspecting related documents. This manual approach is impractical when handling hundreds of different topics. To scale to large projects (thousands of lines of code and multiple releases), we need automatic assistance to determine the topic label. We propose a method of interpreting and labelling topics, and their related artifacts, that is generalizable and relevant to multiple projects.

In our previous work we dealt with topic trends, which are topics that recur over time [13]. We observed that topic trends were often non-functional requirements (NFRs). NFRs have the property of being cross-domain and widely applicable. In this sense, they are useful abstractions for developer conversations about different software projects.

In general, the mining of software artifacts tends to be very project specific, yet NFRs are not. There is a series of standards on NFRs, such as [14], specifically intended to apply to projects of varying types. These standards indicate our approach is possible. In particular, we use software quality models to generalize topics across projects. This allows us to mine software systems with the intention of comparing their NFR-related development topics.

There are many practical applications of *labelled topic extraction*. These methods are useful for providing abstractions of development activities and artifacts. Particularly in large projects, summarization of developer activities is essential to maintenance. Rather than looking for specific concepts in a project [20], we are interested in understanding a software project relative to common properties of all software. We can improve project dashboards, used by managers, by showing both the current topics of development as well those that relate to domain-independent taxonomies, such as NFRs. Some quality assurance questions can be addressed by seeking evidence within repositories that NFR-related topics are being addressed. Topic modelling can be used to check if development is on track and focused. For outside users of the software, our technique can provide insight into that company's development practices in order to answer questions such as, "Are they focused on usability?" Within the organization *labelled topic extraction* can act as a historical sanity check, used to avoid *confirmation bias*, which is the difference between what people think happened and what actually happened. Some software releases, for instance, are poorly received since they are quite buggy. Our technique would help explain why this was the case, since it can identify NFRs which are dominating developer attention.

In this paper, we describe *labelled topic extraction*. It addresses two gaps in the topic mining literature:

1. Topic mining of software has been limited to one project at a time. This is because traditional topic mining techniques are specific to a particular data-set. *Labelled topic extraction* allows for comparisons *between* projects.

2. Topic modeling creates word lists that require interpretation by the user to assign meaning. Like (1), this means that it is difficult to discuss results independent of the project context. Our technique automatically assigns labels across projects.

This paper first introduces labelled topic extraction, and then uses that technique to analyze development activity in two large-scale examples. We begin by describing how we generated our data (Sections 2.1 and 2.2). We show that labels with their topics can be learned and used to classify other data-sets, either without training (Section 2.3), or

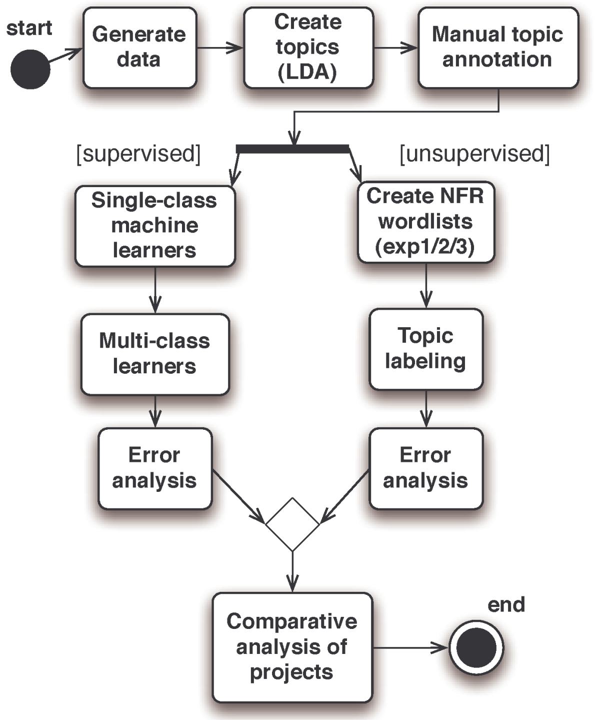


Figure 1: Research methodology process view.

with training (Section 2.4). We then present visualizations of named topics and their trends over time to aid communication and analysis. We use an exploratory case study of two open source database systems to show how named topics can be compared between projects (Section 3). The paper concludes with a discussion of limitations (Section 4) and related work (Section 5).

2. STUDY DESIGN AND EXECUTION

Figure 1 gives an outline of the methodology followed. We began by gathering source data. We then applied unsupervised and supervised learning techniques to identify topics. These topics were used to analyze the role of non-functional requirements in software maintenance.

2.1 Generating the data

To evaluate our approach, we sought candidate systems that were mature projects and had openly accessible source control repositories. We selected systems from the same application domain, so the functional requirements would be broadly similar. We selected MySQL 3.23 and MaxDB 7.500 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 and MySQL 3.23 was released in early 2001. MySQL contains 320 thousand lines of C and C++ source code. Choosing an older version of these projects allowed us to focus on projects which have moved into the maintenance phase of the software life-cycle.

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

For each project, we used source control commit comments, the messages that programmers write when they commit revisions to a source control repository. This is the most readily accessible source of project interactions for outside researchers; bug trackers and email-list archives were not available for both projects. 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 *the* and *at*. Stemming was performed in the later stages of our analysis.

For the commit message data-sets of each project, we created an XML file which separated commits into 30 day periods. This size of period, 30 days, is smaller than the time between minor releases but large enough for there to be sufficient commits to analyze [13]. For each 30 day period of each project, we input the messages of that period into Latent Dirichlet Allocation (LDA), a topic analysis algorithm [2], and recorded the topics the algorithm returned.

A topic analysis tool like LDA will try to find N independent word distributions within the word distributions of all the messages. Linear combinations of these N word distributions are meant to represent and recreate the word distributions of any of the original messages. These N word distributions effectively form topics: cross-cutting collections of words relevant to one or more of our commit messages. LDA extracts topics in an unsupervised manner; the algorithm relies solely on the source data, word distributions of messages, 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 [13]. A topic, in this paper, 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).

We applied Blei's LDA implementation [2] against the word distributions of these commits, and generated lists of topics per period. We 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 [13].

2.2 Generating word lists

Topics are word distributions, lists of words ranked by frequency, that are burdensome to interpret and hard to distinguish and understand. While the topic models themselves are generated automatically, how to interpret these topics is less clear. For example, in Baldi et al. [1], topics are named manually: human experts read the highest-frequency members of a topic and assign a label accordingly. Given the word list *"listener change remove add fire"*, Baldi et al. would assign the label *event-handling*. The labels are reasonable enough, but still require an expert in the field to determine them. We sought to automate the process of naming the topics.

Accordingly, we tried to associate each topic with a label from a list of keywords and related terms. We intersected the words of the topics and the words of our word-lists. We 'named' a topic if any of its words matched any of the word-list's words. A topic could match more than one keyword. We used several different sets of word-lists for comparison, which we refer to as exp1, exp2, exp3 in the text which follows.

Our first word-list set, exp1, was generated using the ontology described in Kayed et al. [15]. 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 [14]. ISO9126 describes six high-level non-functional requirements (listed in Table 1). There is some debate about the significance and importance of the terms in this model. However, 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. We use these qualities later on as classes in supervised labelling. 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 [9], an English-language "lexical database" that contains semantic relations between words, including common related forms (similar to word stemming), meronymy and synonymy. We then added Boehm's 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. [23]. Finally, we analyzed two mailing lists from another software project to expand our set with domain-specific terms. For example, we add the term "performance" to the synonyms for *efficiency*, since this term occurs in most mail messages that discuss efficiency.

For the third set of word-lists, exp3, we extended the word-lists 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. For example, the label *maintainability* is associated with words *ease* and *ownership*.

Creating a validation corpus — For MySQL 3.23 and MaxDB 7.500, we manually annotated each extracted topic in each period with the same NFR 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 the topic being annotated. 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 the data-set for our supervised machine 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: NFRs and associated signifiers – exp2

Project	Measure	exp1	exp2	exp3
MaxDB 7.500	Named Topics	281	125	328
MaxDB 7.500	Unnamed Topics	219	375	172
MaxDB 7.500	Total Topics	500	500	500
MySQL 3.23	Named Topics	524	273	773
MySQL 3.23	Unnamed Topics	476	727	227
MySQL 3.23	Total Topics	1000	1000	1000

Table 2: Automatic topic labelling for MaxDB and MySQL

based topic classification.

2.3 Unsupervised labelling

Using our three word lists, we labeled our topics where there was a match between a word in the list and the same word in the topic. A *named topic* is a topic with a matching label. There are roughly 20 times the number of topics as there are periods per project. This is because we told LDA to extract 20 topics per period. All experiments were run on MaxDB 7.500 and MySQL 3.23 data. Table 2 shows how many topics were labelled for MaxDB and MySQL.

For exp1, our best performing labels, those 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 terms were *verifiability* with *traceability*, and *testability* with *correctness* (76 and 62 matches, respectively).

For exp2, there are more unnamed topics than exp1. Only *reliability* produces a lot of matches, mostly with the word ‘error’. Co-occurrence results were poor. This suggests our word lists were overly restrictive.

For exp3, we generally labelled more topics. As we mentioned, the word-lists are quite broad, so there are likely to be false-positives (discussed below). We found a high of 265 topics for *usability*, with a low of 44 topics for *maintainability*. Common co-occurrences were *reliability* with *usability*, *efficiency* with *reliability*, and *efficiency* with *usability* (200, 190, and 150 topics in common, respectively).

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 assessed whether unsupervised labels matched the manual annotations. Figure 2 shows our results for MaxDB and MySQL. Our performance is measured using the Receiver Operating Characteristic area-under-curve value [8], abbreviated *ROC*. ROC values provide a score, similar to school letter-grades (A is 0.9, C is 0.6), reflecting how well a particular learner performed for the given 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. A ROC result of 0.5 would be equivalent to a random learner (that is, issuing as many false positives as true positives). The area under the ROC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance.

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 classifier nonetheless performed substantially better than random.

2.4 Supervised labelling

Supervised labelling requires expert analysis of the correct

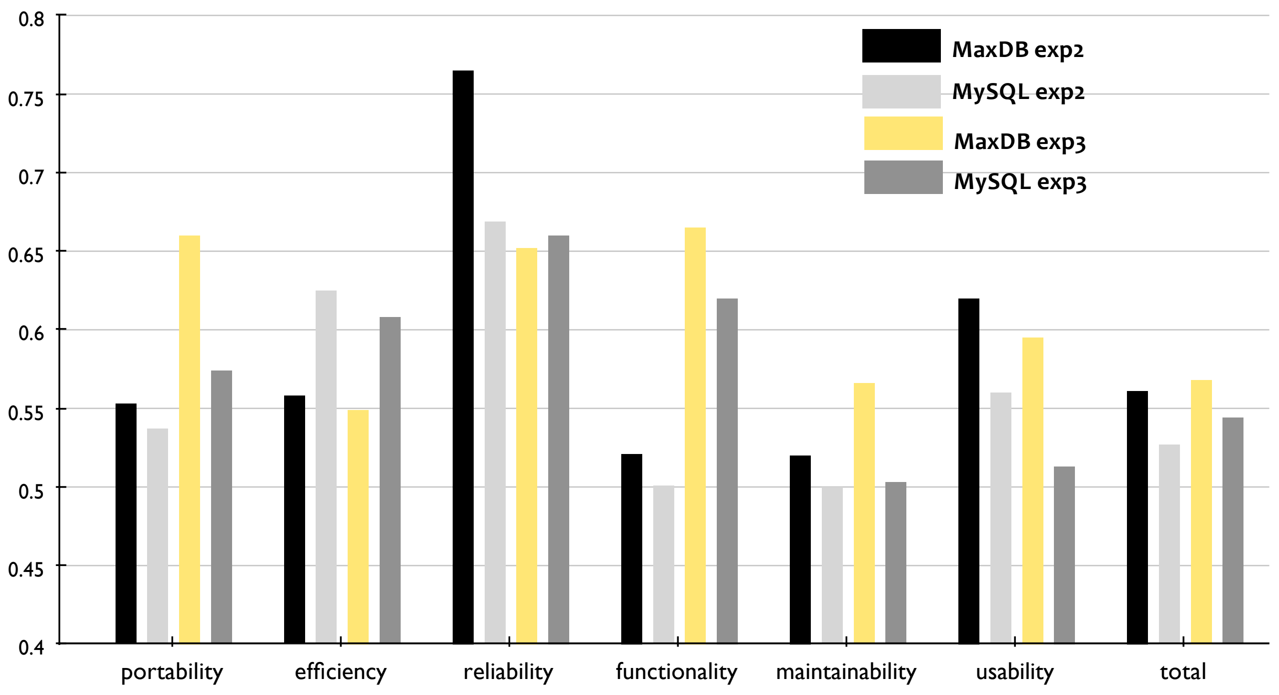


Figure 2: Performance, ROC values (range: 0-1), of unsupervised topic labelling for each NFR and per word-list. This graph shows how well the unsupervised topic labelling matched our manual annotations.

class/label to assign to a topic. In our approach, we use the top-level NFRs in the ISO 9126 standard [14] for our classes, but other taxonomies are also applicable.

We used a suite of supervised classifiers, WEKA [11]. WEKA includes machine learning tools such as support vector machines and Bayes-nets. We also used the multi-labelling add-on for WEKA, Mulan [30]. 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 [17], a common technique for evaluating machine learners. The original data is partitioned randomly into ten sub-samples. Each sample is used to test against a training set composed of the nine other samples. We have reported these results below.

Analysis of the supervised labelling – Because our data-set was of word counts we expected Bayesian techniques, often used in spam filtering, to perform well. We tried other learners that WEKA [11] provides: rule learners, decision tree learners, vector space learners, and support vector machines. Figure 3 shows the performance of the best performing learner per label. We considered the best learner for a label to be the learner which had the highest ROC value for that label.

Figure 3 shows that MaxDB and MySQL have quite different results, as the ROC values for *reliability* and *functionality* seem swapped between projects. For both projects Bayesian techniques did the best out of a wide variety of machine learners tested. Our best learners, Discriminative Multinomial Naive Bayes, Naive Bayes and Multinomial Naive Bayes 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 less-frequently occurring a label, the harder it is to get accurate results, due to the high noise level. Nevertheless, these results are better than our previous word-list 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. The next section (2.5) addresses the issue of a lack of independence and correlation between labels. In the next section we will evaluate how well these learners perform together.

2.5 Applying multiple labels to topics

As noted in Section 2.1, each topic in our data-set can be composed of zero, one, or more NFRs. For example, a commit message might address *reliability* in the context of *efficiency*, or make a *maintainability* improvement in the source code that related to *usability*. However, traditional machine learning techniques, such as Naive Bayes, can only map topics to a single class. The Mulan [30] library encapsulates several different multi-label machine learners which can label elements with multiple labels. Mulan also includes methods for determining the performance of such techniques.

Two perspectives to evaluate multi-label learners are with micro or macro measurements (used in Figure 4). Macro

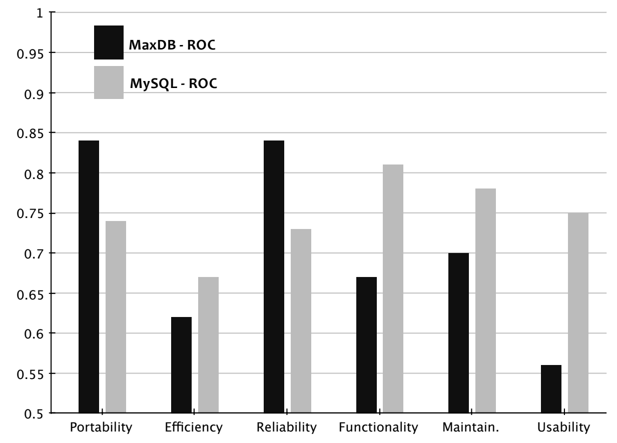


Figure 3: ROC value for the best learner per label for MaxDB and MySQL. Values range from 0–1.

measurements are aggregated at a class or label level (per class) while micro measurements are at the element level (per element). 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. For MaxDB, the macro-ROC values are undefined because of poor performance of one of the labels.

Figure 4 presents the results of Mulan's best multi-label learners for our data. Calibrated Label Ranking (CLR) is a learner that builds two layers, the first layer determines if an entity should be labelled, while the second layer determines what labels should be assigned. The Hierarchy Of Multi-label classifiERs (HOMER) and Binary Relevance (BR) act as a hierarchy of learners: BR is flat, while HOMER tries to build a deeper hierarchy to build a more accurate learner [30]. These classifiers performed better than other multi-label classifiers as they have the best micro and macro ROC scores. The multi-label and single-label learners had similar performance: for MySQL, BR and NaiveBayes had similar macro-ROC scores of 0.74.

2.6 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*, likely because concurrent code is prone to bugs such as race conditions. *Configuration management* and *source control* related changes appeared often; 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 were applied to files. In these cases we assigned the *None* class.

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 addresses correctness or another quality. Despite this difference, the name of the module could fool our learners into believing the change was always about *efficiency.* In these cases the advantages of tailoring topic names to specific project terminologies are more clear. Project specific word-lists would avoid automated mistakes due to the names of entities and modules of a software project.

2.7 Summary of techniques

While an unsupervised technique such as LDA is appealing in its lack of human intervention, and thus lower effort, supervised learners have the advantage of domain knowledge, which typically means improved results. Our manual annotations were fairly quick to do, taking only a few minutes per 30 day period.

Very rarely did `exp2` and `exp3` (naive word matching) ever perform as well as the supervised machine learners. For MaxDB, *reliability* 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, which made it difficult to detect in any case.

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

3. UNDERSTANDING SOFTWARE MAINTENANCE ACTIVITIES

Since our ROC values produced acceptable results, we now turn to *applying* those techniques to understand software maintenance activities. We evaluated two research questions:

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

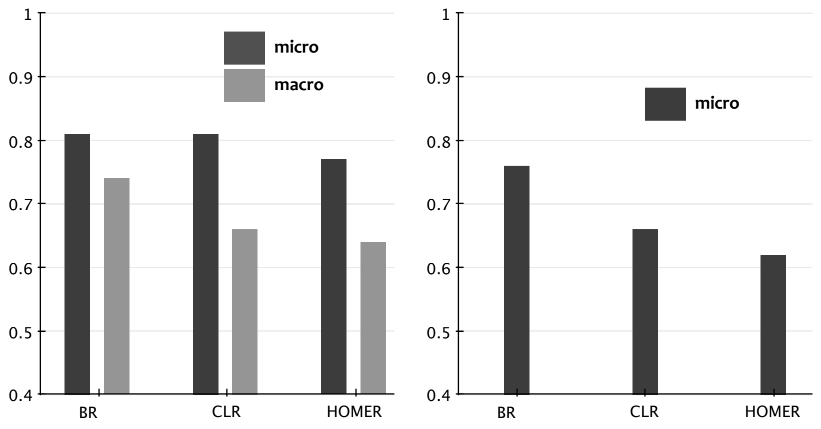
2. Do the different projects differ in their relative topic interest? Is a particular quality more important to one project than the other projects?

3.1 Timeline analysis

Figures 5a and 5b show the temporal patterns of label frequencies. There are two measures represented. One, the relative frequency, shown in the grey boxes, represents the number of labels with that NFR in that period, relative to the maximum number of topics assigned to that NFR. The second, absolute frequency, compares the number of topics labelled with that NFR per period relative to the maximum number of labelled topics overall, for that project. The topmost row in each diagram is reserved for historical events of note. We refer to these occurrences in the discussion which follows.

In Figure 5a, we see that the NFRs *functionality*, *portability* and *maintainability* contain more labeled topics, since these NFRs have been more intensely shaded. There are more `None` labels in MaxDB because a number of topics were to do with code cleanup or automated checking using tools like `sutcheck`, a tool-specific to the development process at MaxDB.

As discussed earlier, the top row in each figure shows key events for each project. *Labelled topic extraction* can pick



(a) MySQL (b) MaxDB

Figure 4: MySQL and MaxDB macro (grey) and micro (black) ROC results per multi-label learner. Possible values range from 0–1.

out the underlying NFR activity behind these events. For example, both projects show a high number of NFRs recognized at the first period of analysis. This is due to our window choice: we deliberately targeted our analysis to when MySQL 3.23 was first announced and when MaxDB 7.500 was first announced. For MaxDB, version 7.5.00 was released in December of 2003. From analyzing the mailing list for external validation, we know that release 7.5.00.23 saw the development of PHP interfaces, possibly accounting for the simultaneous increase in the *portability* NFR at the same time. The gap in MaxDB (Figure 5b) is due to a shift in development focus (from February 2005 to Jun 2005) to MaxDB 7.6, which is released in June 2005.

For MySQL (Figure 5a), we similarly validated our NFR patterns with external mailing list data. This release of MySQL was the first to be licenced under the GPL. Version 3.23.31 (January, 2001) was the production release (non-beta), and we see a flurry of topics labelled with *functionality* and *maintainability*. After this point, this version enters the maintenance phase of its life-cycle. In May 2001, there is an increase in the number of topics labelled with *portability*. This might be related to release 3.23.38, which focused on Windows compatibility. Similarly, in August, 2002, both *functionality* and *portability* are frequent, and mailing list data suggests this is related to the release of version 3.23.52, a general bug fix with a focus on security (a component of the *functionality* NFR in the ISO9126 model). After this point, efforts shift to the newer releases (4.0, 4.1, 5.0).

This analysis allows to address the questions raised in Section 2.4:

Do label frequencies change over time? – In both projects the frequencies generally decreased with age. However, there are variations within our NFR labels. In MySQL, *usability* and *efficiency* do not appear very often in topics. A proportionately smaller number of commits addressed these NFRs. As discussed in the previous section, certain peaks in topic numbers coincide with a particular emphasis from the development team on issues such as new releases or bug fixes. This suggests that maintenance activity is not necessarily strictly decreasing with time, but rather episodic and responsive to outside stimuli. In MaxDB, we can observe that *Maintainability* topics became more prevalent as MaxDB matures. This is likely due to our analysis timeframe for MaxDB being shorter than the timeframe for the MySQL product.

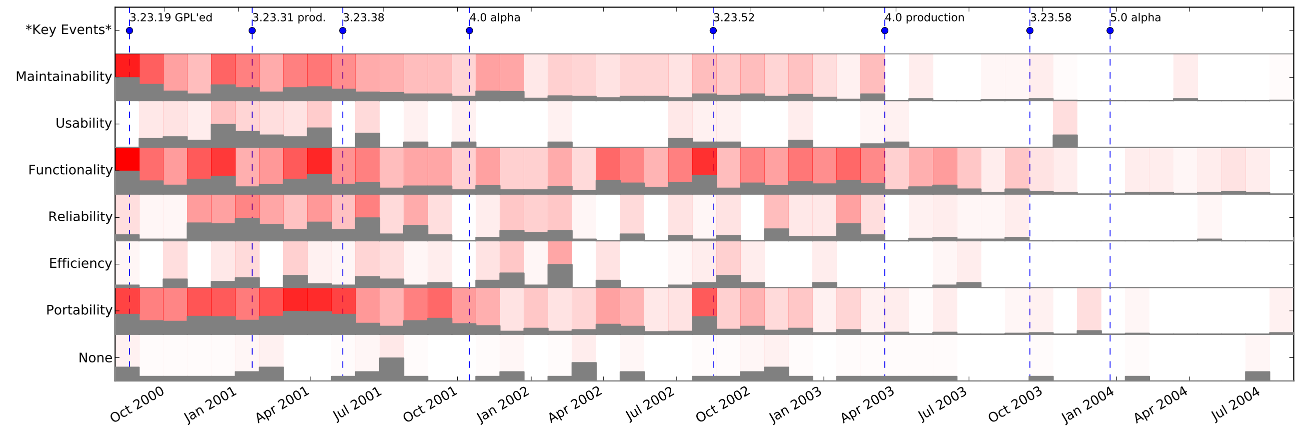
Do the different projects differ in their relative topic interest? – Yes. MySQL 3.23 had proportionally more topics labelled *functionality*, while MaxDB had proportionally more *efficiency* related topics. MaxDB was a very mature release 'donated' to the open-source community. MySQL, on the other hand, was in its relative infancy. Security problems were more common (security is a component of 'functionality' in the ISO 9126 model). In both cases *portability* was a constant maintenance concern and was prevalent throughout the entire lifetime of the projects.

4. DISCUSSION

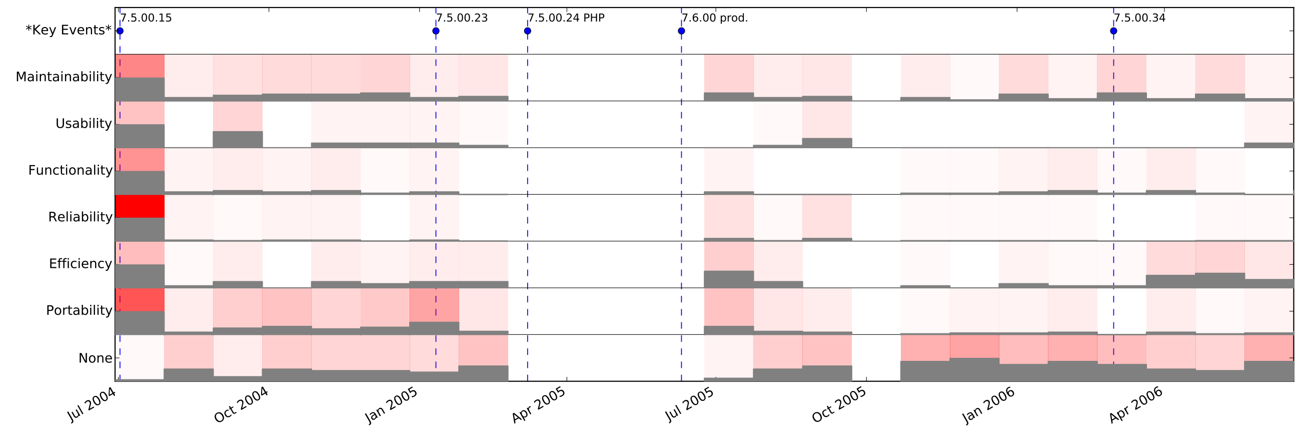
The unsupervised labelling had difficulty distinguishing between common labels and infrequent labels. The learners would occasionally mislabel a topic deserving of an infrequent label with a more common label. The word-lists for *correctness* tended to be too lengthy, non-specific and broad, especially if WordNet words were used, as correctness is a loosely defined concept in common parlance.

With ROC values ranging from 0.6 to 0.8 we can see there is promise in these methods. exp2 and exp3 both indicate that static information can be used to help label topics without any training whatsoever. MySQL and MaxDB's machine learners made some decisions based off a few shared words: bug, code, compiler, database, HP UX, delete, memory, missing, problems, removed, add, added, changed, problem, and test. Adding these words to the word-lists of exp2 and exp3 could improve performance while ensuring they were only domain specific.

If the techniques used in exp2 and exp3 were combined with the supervised techniques we could reduce the training effort by boosting training sets with topics classified with the unsupervised techniques. Both Naive Bayesian learners and the word-list approaches were computationally efficient. These results are promising because they indicate that these techniques are accurate enough to be useful while still main-



(a) MySQL 3.23



(b) MaxDB 7.500

Figure 5: NFR label per period. Each box represents a 30-day period. Grid cell intensity (saturation) is mapped to label frequency relative to the largest label value of *all* NFRs. Grey bars indicate label frequency relative to *a particular* NFR. Dashed vertical lines relate a project milestone (*Key Event*) to our topic windows.

taining acceptable run-time performance.

4.1 Threats to validity

Our work faced multiple threats to validity and we attempted to address many of them.

Construct validity – we used only commit messages rather than mail or bug tracker messages. Conversely, construct validity was addressed in our validation as we used the files, topics and revisions to annotate the commit messages. We used mailing-lists to verify the purpose behind some events. We also chose our taxonomy and the data to study. We did not show these results to the stakeholders.

Internal validity – includes inter-rater reliability and our lack of rival explanation building. We improved internal validity by trying to correlate and explain the behaviours observed in the analysis with the historical records of the projects. We did not attempt to match our results to any particular model or theory.

External validity was addressed by our use of multiple case studies, but there are still issues. 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.

Reliability was addressed by each annotator following the same protocol and using the same annotations. Unfortunately only two annotators were used; their annotations could be biased as we did not analyze for inter-rater reliability.

5. RELATED WORK

The idea of extracting higher-level 'concerns', also known as 'concepts', 'aspects' or 'requirements', has been approached from documentation-based and repository-based perspectives.

Cleland-Huang and her colleagues published work on mining requirements documents for non-functional requirements (NFR) (quality requirements) [6]. Their approach was similar to ours, as they mined keywords from their own NFR catalogues [5]. They managed a 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. Our worked differed because we had a more comprehensive set of terms based on the taxonomy we chose. We also wanted a common taxonomy that allows us to make cross-project comparison; this was not one of their objectives. The objective of Cleland-Huang's study was to identify new NFRs for system development, yet our objective was to recover those latent NFRs from commit-log messages of the project. Another difference was that our data was far less structured than the requirements documents used in their study.

In the same vein, Mockus and Votta [26] studied a large-scale industrial change-tracking system. They also leveraged WordNet, but only for word roots as they felt the synonyms would be non-specific and cause errors. Mockus et al. had access to system developers, with whom they could validate their labels. Since we try to bridge different organizations, these kind of interviews are infeasible (particularly in the distributed world of open-source software).

The other approach is to extract concerns from software repositories. Marcus et al. [20] describe their use of Latent Semantic Indexing to identify commonly occurring concerns for software maintenance. ConcernLines [28] shows tag occurrence using colour intensity. They mined developer created tags in order to analyze the evolution 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. [25] conducted an empirical study of Eclipse, a popular Java IDE, to verify the claims of Lehman [19]. 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. [24] 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, and does not leverage machine learning strategies.

Massey [21] and Scacchi [27] looked at the topic of requirements in open-source software. Their work discusses the source of the requirements and how they are used in the development process. German [10] looked at GNOME specifically, and listed several sources for requirements: leader interest, mimicry, brainstorming, and prototypes.

Hindle et al. [12] examined release patterns in OSS. They showed that there is a difference between projects regarding maintenance techniques. This supports our result that software qualities are not discussed with the same frequency across projects.

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented a cross-project data mining technique, *labelled topic extraction*. The technique leveraged a non-functional requirements (NFRs) taxonomy to mine and label latent software development topics. Our contributions include:

- We provided an unsupervised method of topic labelling (word-lists);
- We demonstrated a supervised method of labelling topics by a single NFRs (machine learning);
- We demonstrated a supervised method of labelling topics by multiple NFRs (multi-label machine learning);
- We provided a method of cross-project analysis via topic labelling;
- We demonstrated a method of visualizing these topics across time, which we used to analyze maintenance activities.

We validated our topic labelling techniques using multiple experiments. We first conducted unsupervised labelling using word-lists. Our next approach was supervised, using single-label and multi-label learners. Both kinds of learners performed well with area under the ROC curve values between 0.6 and 0.8.

These labelling techniques allowed us to investigate the occurrence of non-functional requirements in our projects. We showed that NFRs are often trending topics; that NFRs are quite common in developer topics; and that there are efficient methods of automating topic labelling. Although it may seem like common knowledge that maintenance is a concern as a project matures, our technique was able to show this independently.

Previous topic analysis research produced topics that needed to be manually labelled, but topics without labels are difficult to interpret. To improve this, we leveraged software engineering standards to produce a method of domain specific topic labelling. Since the word-list technique is not project specific, it can be used to compare multiple projects. We demonstrated multiple supervised and unsupervised methods of topic labelling, including multi-label learners. We provided a case study and visualization of NFR related topics in MaxDB 7.500 and MySQL 3.23. During the case study we manually labelled hundreds of topics by hand and then used these annotations to validate the positive performance of topic labelling techniques. The implication of our paper is that topics with labels are more useful for analyzing software development activities than are unlabelled topics. To furnish stakeholders with labelled topics, our technique provides many methods of automatic topic labelling.

There are several avenues of further investigation. More external validation would be useful. Although we validated our comparisons using a mailing list for each project, interviews with developers would provide more detail. We also think multi-label learning techniques, although in their infancy, are crucial in understanding cross-cutting concerns such as NFRs. We want to leverage different kinds of artifacts to discover threads of NFR-related discussions that occur between multiple kinds of artifacts. Finally, we would like to extend this analysis to other domains, to see what patterns might occur in, for example, a consumer-facing software product.

APPENDIX

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

A. 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.

[8] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006.

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

[10] D. M. German. The GNOME project: a case study of open source, global software development. *Software Process: Improvement and Practice*, 8(4):201–215, 2003.

[11] 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.

[12] A. Hindle, M. W. Godfrey, and R. C. Holt. Release Pattern Discovery via Partitioning: Methodology and Case Study. In *International Workshop on Mining Software Repositories at ICSE*, pages 19–27, Minneapolis, MN, May 2007.

[13] 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.

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

[15] 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.

[16] M. Kersten and G. Murphy. Mylar: a degree-of-interest model for IDEs. In *International Conference on Aspect-oriented Software Development*, pages 159–168, Chicago, March 2005.

[17] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference On Artificial Intelligence*, pages 1137–1143, Toronto, 1995.

[18] a. Kuhn, S. Ducasse, and T. Girba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243, March 2007.

[19] 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.

[20] 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.

[21] B. Massey. Where Do Open Source Requirements Come From (And What Should We Do About It)? In *Workshop on Open source software engineering at ICSE*, Orlando, FL, USA, 2002.

[22] T. McCabe. A complexity measure. *IEEE Transactions on software Engineering*, 2(4):308–320, 1976.

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

[24] 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.

[25] 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.

[26] 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.

[27] W. Scacchi, C. Jensen, J. Noll, and M. Elliott. Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes. In *International Conference on Open Source Systems*, volume 1, pages 1—-8, Genoa, Italy, July 2005.

[28] 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.

[29] C. Treude and M.-A. Storey. Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In *International Conference on Software Engineering*, pages 365–374, Cape Town, South Africa, 2010. ACM.

[30] 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.