# Automated Topic Naming to Support Cross-project Analysis of Software Maintenance Activities

Abram Hindle
Dept. of Computer Science
University of California, Davis
Davis, CA, USA
abram@softwareprocess.es

Neil A. Ernst
Dept. 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

John Mylopoulos
Dept. Information Eng. and Computer Science
University of Trento
Trento, ITALY
jm@disi.unitn.it

## ABSTRACT

Researchers have employed a variety of techniques to extract underlying topics that relate to software development artifacts. Typically, these techniques use semi-unsupervised machine-learning algorithms to suggest candidate word-lists. However, word-lists are difficult to interpret in the absence of meaningful summary labels. Current topic modeling techniques assume manual labelling and do not use domain-specific knowledge to improve, contextualize, or describe results for the developers. We propose a solution: *automated 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 from 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. The case studies show that labelled topic extraction can produce 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**]: Management—*Lifecycle*; D.2.1 [**Software Engineering**]: Requirements/Specifications—*Tools*

## General Terms

Human Factors, Management, Measurement

## Keywords

Topic analysis, LDA, non-functional requirements

*MSR*'11, May 21–22, 2011, Waikiki, Honolulu, HI, USA

## 1. INTRODUCTION

A key problem for practising software maintainers is gaining an understanding of *why* a system has evolved the way it has. This is different than *how* a system has evolved, because the behaviour of change itself is the *how.* Looking back on streams of artifacts scattered across different repositories, inferring what activities were performed, when, and for what reasons, is hard to do without expert advice from the developers involved. In this work we seek to provide a method of automatically labelling development topics extracted from commit logs.

Topic modeling is a machine learning technique which creates multinomial distributions of words extracted from a text corpus. This technique infers the hidden structure of a corpus using posterior inference: the probability of the hidden structure given the data. Topic models are useful in software maintenance because they summarize the key concepts in a corpus, such as source code, commit comments, or mailing-list messages, by identifying statistically co-occurring words. Among other uses, it can quickly give developers an overview of where significant activity has occurred, and gives managers or maintainers a sense of project history.

While machine learning techniques can automatically identify clumps of commonly recurring terms, devising an appropriate summary label for each clump/topic is harder. A given topic extracted from a set of commit logs might consist of the following terms: *"listener change remove add fire".* This topic might reasonably be labelled as "*event handling*" by a developer who understands the domain well, despite the fact that this label does not appear in the word list itself. Current approaches to topic labelling rely on manual intervention by human experts, and also are limited to project-specific topic labels. In this paper, we introduce *labelled topic extraction*, an approach to topic labelling that creates labels automatically that are project independent.

In general, the fruits of mining software artifacts are often project specific and hard to generalize. However, in our previous work we investigated *topic trends* — that is, topics that recur over time — we observed that topic trends often corresponded to non-functional requirements (NFRs) [11], which is further emphasized in this paper due to the large numbers of NFR labelled topics. This is encouraging, as

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. Furthermore, there is a series of standards on NFRs, such as ISO9126 [12], that are specifically intended to apply to projects of varying types; this suggests that our goal of trying to extract NFR-related development topics, such as those related to software quality models, holds promise.

Concrete applications of topics and *labelled topic extraction* range from project dashboards to annotating software artifacts such as revisions and bug reports with NFR-related tags. Project dashboard [9] are typically employed by managers and are used to provide quick summaries of the effort put into a software project. In this case, labelled topic extraction would allow managers to track effort related to NFR topics, such as portability. These techniques also allow for the annotation of commit comments and other software artifacts with NFRs. This would enable querying of bug reports and artifacts by relevant NFRs. For instance a manager can confirm if their developers were focused on *usability* by looking for *usability*-relevant revisions and bug reports.

In this paper, we describe *automated 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. *Automated 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, or with some initial training, assigns labels across projects.

This paper makes the following contributions:

- introduces the concept of labelled topic extraction, using a non-functional requirements (NFR) taxonomy for our labels;

- evaluates three kinds of automatic topic labelling methods: semi-unsupervised labelling of topics (word-lists), supervised labelling of topics with a single NFR (machine learning), and supervised labelling of topics with multiple NFRs (multi-label machine learning);

- provides a method of cross-project analysis via topic labelling; and applies these techniques to visualize NFRs over time, and to analyze maintenance activities.

We begin by discussing related work in Section 2. Next, we describe how we generated our data (Section 3.1). For semi-unsupervised classification (Section 3.2), we begin by creating word-lists to signify when a topic matches an NFR label. We then apply our classifier and analyze the results. In Section 3.3, we manually annotate the topics, and use those annotations as training data for supervised classification. To demonstrate an application of labelled topic extraction, we use an exploratory case study of two open source database systems to show how named topics can be compared between projects (Section 4). The paper concludes with a discussion of limitations (Section 5), and future work.

## 2. PREVIOUS WORK

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

Cleland-Huang and her colleagues have investigated mining requirements documents for non-functional requirements (NFR) (quality requirements) [5]. Their approach is similar to ours, as they mined keywords from NFR catalogues. They differ because they mine requirements documents where as we mine revisions. They demonstrated 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 research differs because we use a more comprehensive set of terms based on the taxonomy we chose. Another difference is that we make cross-project comparisons instead of focusing on a single project.

Similarly, Mockus and Votta [18] studied a large-scale industrial change-tracking system. Mockus and Votta leveraged WordNet [8], an English-language "lexical database" that contains semantic relations between words, including common related forms (similar to word stemming), meronymy and synonymy. They used WordNet for word roots as they felt the synonyms would be non-specific and cause errors. Mockus et al. validated their labels with system developers. Since we study multiple projects, instead of a single project, these kind of interviews are somewhat infeasible (particularly in the distributed world of open-source software).

Another approach is to extract concerns from software repositories. Marcus et al. [15] use Latent Semantic Indexing (LSI) to identify commonly occurring concerns for software maintenance. The concerns are given by the user, and LSI is used to retrieve them from a corpus. Topic modelling generates topics that are independent of a user query, and relate only to word frequencies in the corpus. With ConcernLines, Treude et al. [19] show tag occurrence using colour and intensity. They mine 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.

In Baldi et al. [2], topics are named manually: human experts read the highest-frequency members of a topic and assign a label accordingly. As discussed earlier, given the topic *"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. Furthermore, these labels are project-specific, because they are generated from the data of that project. E.g., we might have a label called *'Oracle'* in the MySQL case, since Oracle owns MySQL. Our approach differs: first of all, we automate the process of naming the topics; secondly, we label topics with project-independent terms, in order to permit cross-project comparison.

Mei et al. [17] 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, but we do not generate phrases from the topics.

Finally, in Ernst et al. [6], we describe our earlier project, similar to this, that identifies changes in quality requirements in GNOME software projects; this approach was more

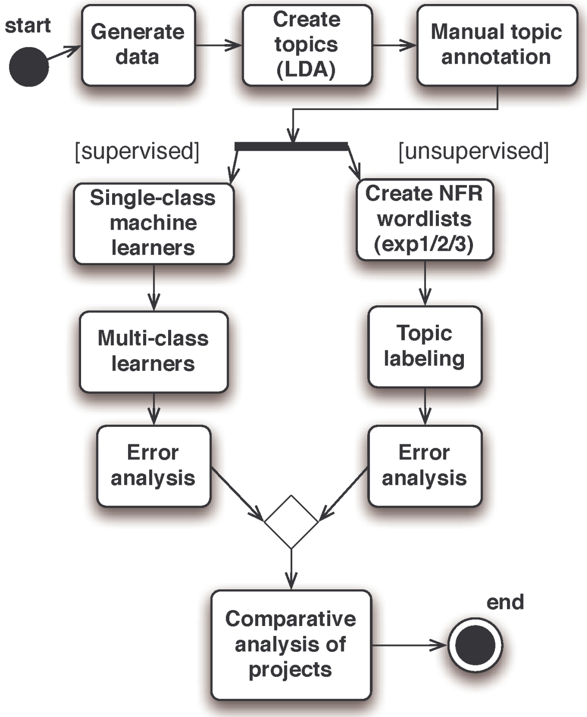


**Figure 1: Research methodology process view.**

exploratory, had less validation, uses different word-lists, solely uses text-matching, and does not leverage machine learning strategies. Hindle et al. [11] propose a windowed method of topic analysis that we extend with labelled topics, NFRs and new visualizations.

# 3. STUDY DESIGN AND EXECUTION

Figure 1 gives an outline of our methodology. We began by gathering source data and creating topic models. For semi-unsupervised labelling, we generated three sets of word-lists as signifiers for NFRs. In supervised learning, we trained our data with manual annotations in order to match topics with NFRs. Finally, these topics were used to analyze the role of NFRs in software maintenance.

## 3.1 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, to control for differences in functional, rather than non-functional, requirements. 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 over 940, 000 lines of C source code[1]. The MySQL project started in 1994 and MySQL 3.23 was released in early 2001. MySQL contains 320, 000 lines of C and C++ source code. We explicitly chose older versions of mature projects from a stable problem domain to increase the likelihood that we would encounter primarily maintenance activities in our studies.

For each project, we used source control commit comments, the messages that programmers write when they commit revisions to a source control repository. Most commits we observed had commit comments. Commit comments are often studied by researchers, as they are the most readily accessible source of project interactions, and developers are often required to create them by the repository mechanism (e.g., Git). Additionally, relying only on commit comments makes our approach more generalizable, as we do not assume the presence of other artifact corpora. An example of a typical commit message, from MySQL, 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 minus stop-words such as "*the*" and "*at*".

For the commit message data-sets of each project, we created an XML file that separated commits into 30 day periods. We chose a period size of 30 days as it is smaller than the time between minor releases but large enough for there to be sufficient commits to analyze [11]. For each 30 day period of each project, we input the messages of that period into Latent Dirichlet Allocation (LDA), a topic analysis algorithm [3], and recorded the topics the algorithm extracted.

A topic analysis tool such as LDA will try to find $N$ independent word distributions within the word distributions of all input 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 and 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 [11]. For this paper, a topic represents a word distribution generated from a group of commit log comments which are related by their content.

We applied Blei's LDA implementation [3] 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 dilutes the results and creates indistinct topics [11].

### 3.1.1 The High-level Labels

To facilitate cross-project comparison, we used a taxonomy of NFRs. This taxonomy is based on the ISO quality model, ISO9126 [12]. ISO9126 describes six high-level NFRs: maintainability, functionality, portability, efficiency, usability, and reliability [2]. We claim that these NFRs are maintenance concerns (to varying degrees) in all software projects, and are therefore well-suited for comparisons between projects.

[1] generated using David A. Wheeler's *SLOCCount*, http://dwheeler.com/sloccount.

[2] While there may be lingering debate in some circles about these terms, an ISO standard seems like a reasonable starting point for our work.

### 3.1.2 Creating a Validation Corpus

To evaluate both semi-unsupervised and supervised classification, we created a validation set of manually labelled topics. For MySQL 3.23 and MaxDB 7.500, the annotators (the first two authors) annotated each extracted topic in each period with the six NFR labels listed above. Annotators did not annotate each other's annotations, but some brief inspection of annotations was used to confirm that the annotators were acting similarly. We looked at each period's topics, and assessed what the data — consisting of the frequency-weighted word lists and messages — suggested was the label for that topic. 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.* Given the top-level annotations of *none*, *portability*, *efficiency*, *reliability*, *functionality*, *usability*, and *maintainability*, the annotators annotated each topic with the relevant label. Sometimes they used finer-grained annotations that would be aggregated up to one of these higher-level labels.

We validate classification performance using the Receiver Operating Characteristic area-under-curve value [7], abbreviated *ROC*, and the F-measure, which is the harmonic mean of precision and recall, i.e., $2 * (P * R)/(P + R)$.

ROC values provide a score 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 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. We consider our labelling classifiers acceptable if they outperform a random classifier (0.5).

## 3.2 Semi-unsupervised Labelling

In this section we describe how to label topics based on dictionaries mined from sources external to the projects.

### 3.2.1 Generating Word Lists

In order to automatically label each topic with one of the six high-level NFRs, we associate each NFR with a list of keywords (*word-lists*, in our parlance). These word-lists were determined a priori and were not extracted from the projects themselves. We intersected the words of the topics and the words of our word-lists. We "labelled" a topic if any of its words matched any of the word-list's words. A topic could match more than one NFR. We used several different sets of word-lists for comparison, which we refer to as exp1, exp2, and exp3 in the text which follows.

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, exp2, uses the ISO9126 taxonomy described above (Section 3.1) to seed the word-lists. The terms from ISO9126 may not capture all words occurring in the topics that are nonetheless associated with one of the NFRs. For example, the term "redundancy" is one we considered to be relevant to discussion of *reliability*, but is not in the standard. We therefore took the NFRs from the ISO9126 and added to them.

To construct these expanded word-lists, we used WordNet [8]. We then added Boehm's software quality model [4], and classified his eleven 'ilities' into their respective ISO9126 NFRs. We did the same for the quality model produced by McCall et al. [16]. We then did a simple random analysis of mailing list messages from an open source project, KDE. If we judged a given message to contain terms that were related to one of the NFRs in ISO9126, we added it to our word-list. This allowed us to expand our word-lists with more software-specific terms.

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 in exp3 are specific enough. For example, the label *maintainability* is associated with words *ease* and *ownership.* In general, as we proceed from word-list in exp1 to that in exp3, our lists become more generic.

### 3.2.2 Automatic Labelled Topic Extraction

Using our three word-lists (exp1, exp2, exp3), we labelled our topics with an NFR where there was a match between a word in the list and the same word somewhere in the distribution of words that constitute the topic. A *named topic* is a topic with a matching NFR label. *Unnamed topics* occur where there is no such match, which may indicate either a lack of precision, or simply that this topic is not associated with non-functional requirements. All experiments were run on MaxDB 7.500 and MySQL 3.23 data. LDA extracted 20 topics per period for each project. This labelling is *semi-unsupervised* because the corpus is not derived from the project being analyzed, and we did not label the project's topics ourselves for a training set. The motivation behind this technique is that because most software often addresses similar issues, thus we can use the domain knowledge of software to label relevant topics.

Table 2 shows how many topics were labelled for MaxDB and MySQL.

For exp1 the labels with the most topics were *correctness* (182 topics) and *testability* (121). We did not see many results for *usability* or *accuracy*, which were associated with fewer than ten topics. We also looked for correlations 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).

| 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 word-lists – exp2**

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

**Table 2: Automatic topic labelling for MaxDB and MySQL**

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 broad, so there are likely to be false-positives (discussed below). The most frequent label, 265 topics, was *usability*, and the least frequent label, 44 topics, was *maintainability*. Common co-occurrences were *reliability* with *usability*, *efficiency* with *reliability*, and *efficiency* with *usability* (200, 190, and 150 topics in common, respectively).

### *3.2.3 Analysis of the Semi-unsupervised Labelling*

For each quality we assessed whether semi-unsupervised labels matched the manual annotations. As described in Section 3.1 we used both ROC and F-1 measures to evaluate the performance of the classification. Figure 2 shows our ROC results for MaxDB and MySQL. We describe F-1 results in the text below.

Because our ground truth annotations were relevant only to ISO9126, we estimate that exp1 had poor performance via the overlap between ISO9126 and the Kayed ontology. For exp1 the F-measures for MaxDB were from 0 to 0.18 with an average of 0.03, and for MySQL were from 0 to 0.16 with an average of 0.05.

For exp2, the average F-measure (macro-F1) for MaxDB was 0.24 with a range 0.091 to 0.37, and 0.16 for MySQL with a range of 0 to 0.41. MaxDB had an average precision and recall of 0.25 and 0.22 while MySQL had 0.41 and 0.10 respectively.

For exp3, the average F-measure (macro-F1) for MaxDB was 0.26 with a range 0.11 to 0.47, and 0.36 for MySQL with a range of 0.10 to 0.65. MaxDB had an average precision and recall of 0.16 and 0.67 while MySQL had 0.3 and 0.48 respectively.

Thus we find that *reliability* and *usability* worked well for MaxDB in exp2 and better in exp3. exp1 performed poorly. 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. Many ROC scores were 0.6 or less, but our classifier still performed substantially better than random.

## 3.3 Supervised Labelling

Supervised labelling requires expert analysis of the correct class/label to assign a label to a topic. In our approach, we use the top-level NFRs in the ISO9126 standard [12] for our classes, but other taxonomies are also applicable.

We used a suite of supervised classifiers, WEKA [10], that includes machine learning tools such as support vector machines and Bayes-nets. We also used the multi-labelling add-on for WEKA, Mulan [20]. Traditional classifiers label topics with a single class, whereas Mulan allows for a mixture of classes per topic, which is what we observed while manually labelling topics. The *features* we used are word counts/occurrence per topic, if a word occurs frequently enough in a topic we consider it a feature of the topic.

To assess the performance of the supervised learners, we did a 10-fold cross-validation [14], a common technique for evaluating machine learners. The original data is partitioned randomly into ten sub-samples. Each sample is used to test

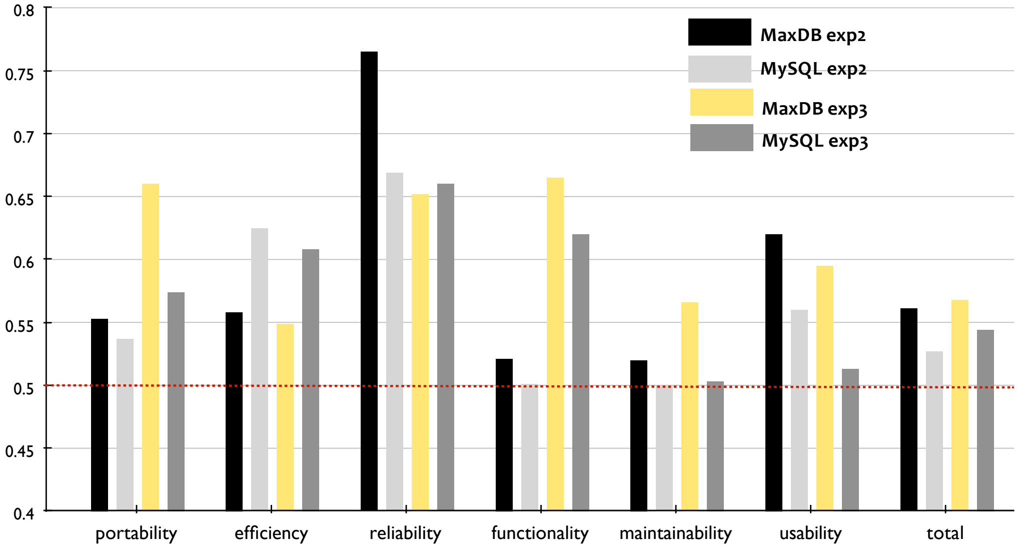


**Figure 2: Performance, ROC values (range: 0–1), of semi-unsupervised topic labelling for each NFR and per word-list. The dashed line indicates the performance of a random classifier. This graph shows how well the semi-unsupervised topic labelling matched our manual annotations.**

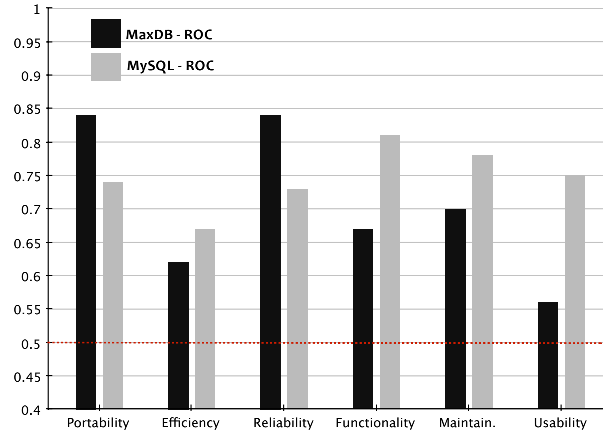


**Figure 3: ROC value for the best learner per label for MaxDB and MySQL. Values range from 0–1. Dashed line indicates the performance of a random classifier.**

against a training set composed of the nine other samples. We have reported these results below.

### 3.3.1 *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 [10] 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: the learner that had the highest ROC value for that label. The best learner is important because one uses a single learner per label. When applying our technique, for each NFR one should select the best learner possible.

Figure 3 shows that MaxDB and MySQL have quite different results, as the ROC values for *reliability* and *functionality* seem swapped between projects.

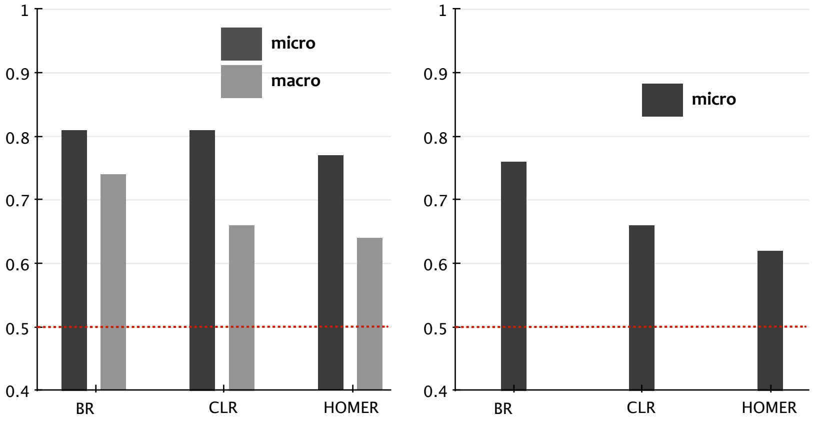
For both projects Bayesian techniques did the best out of a wide variety of machine learners tested. Our best learners, Discriminative Multinomial Naive Bayes, Naive Bayes and Multinomial Naive Bayes are all based on Bayes's theorem and all assume, naively, that the features supplied are independent. 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 range of F-measures for MySQL was 0.21 to 0.77 with an average of 0.48, while MaxDB had a range of 0.17 to 0.61 with an average of 0.39.

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 (3.4) addresses the issue of a lack of independence and correlation between labels using multi-label learners.

## 3.4 Applying Multiple Labels to Topics

As noted in Section 3.1, 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 [20] library encapsulates several different multi-label machine learners which can label



(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. Dashed line indicates the performance of a random classifier.**

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 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 for a more accurate learner [20]. 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 Naive Bayes had similar macro-ROC scores of 0.74.

## 4. UNDERSTANDING SOFTWARE MAINTENANCE ACTIVITIES

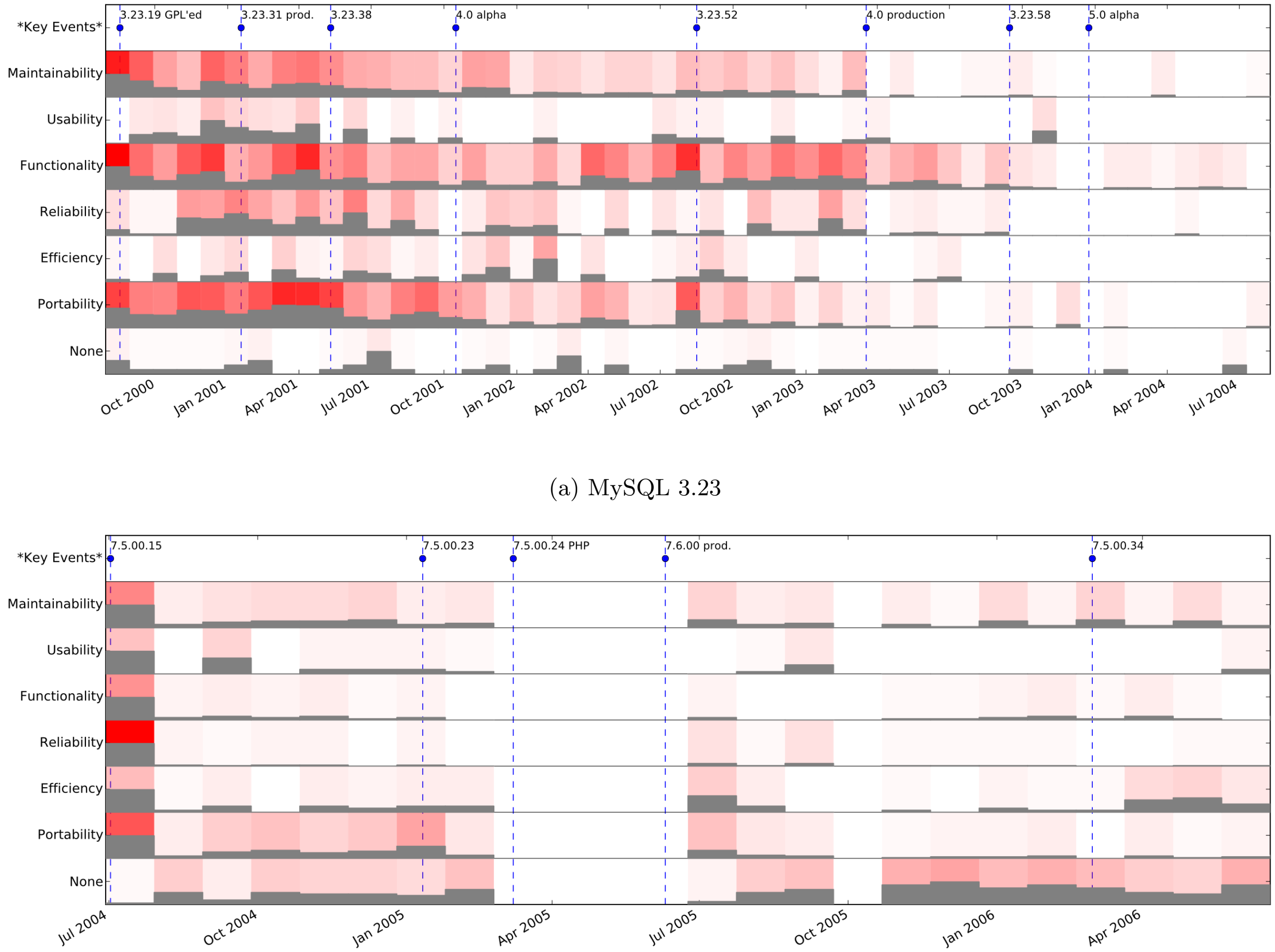
As we mentioned in the introduction, a key issue in software maintenance is understanding *why* a system has evolved the way it has [1]. In this section we demonstrate the value of labelled topic extraction in addressing this issue. Labelled topics address *why* because they show reasons why changes occurred rather than *how*. The *how* of a change is the change itself, the purpose or *why* is what we are after. We investigate the history of the two large-scale database systems that we studied. We use our technique to show the topic of development efforts over time in each project. We motivated our investigation with two research questions:

1. *Do NFR frequencies change over time?* If a particular NFR was of more interest at one point in the life-cycle than another, this suggests that development activity shifted focus. For example, if a developer expected to see a recent focus on *reliability*, but instead *usability* dominated, they might re-prioritize upcoming work items.

2. *Do projects differ in their relative interest in NFRs?* A project manager, especially a systems-manager, would be interested in knowing whether a particular NFR, such as *reliability*, was more important for one project than another. This could be to confirm the initial design goals, or to track the progress on that quarter's objectives. The difference in NFR proportion is interesting because it implies a difference in focus between two projects.

Figures 5a and 5b show the temporal patterns of NFR frequencies. There are two measures represented. One, the relative frequency, shown in the grey histogram boxes, represents the number of topics with that NFR in that period, relative to the maximum number of topics assigned to that NFR. For example, in Figure 5a we see a spike in *portability* and *functionality* frequency in September 2002. The second, absolute frequency, is shown using cell intensity, and compares the number of topics labelled with that NFR per period relative to the maximum number of labelled topics overall. For instance, Figure 5a shows that the NFRs *functionality*, *portability* and *maintainability* contain more labelled topics, since these NFRs have been more intensely shaded; one interesting stream is *efficiency* which shows periodic activity, does this suggest that once maybe efficiency related changes have longer lasting effects? The topmost row in each diagram lists historical events for that project (such as a release).

We analyzed each project's developer mailing list for external validation. We use *labelled topic extraction* to pick 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 both MySQL 3.23 and MaxDB 7.500 were first announced. For MaxDB, version 7.5.00 was released in December of 2003. We know



(a) MySQL 3.23

(b) MaxDB 7.500

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

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 June 2005) to MaxDB 7.6, which is released in June 2005.

The release of MySQL we study (Figure 5a) 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). We now address our research questions:

## 4.1 Do NFR 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. 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 time-frame for MaxDB being shorter than the time-frame for the MySQL product.

## 4.2 Do 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, whereas MySQL was in its relative infancy, and security problems were more common (security is a component of *functionality* in the ISO9126 model). In both cases *portability* was a constant maintenance concern and was prevalent throughout the lifetime of the projects. It may surprise developers how often portability arises as a concern.

# 5. DISCUSSION

## 5.1 Annotation Observations

We found many topics that were not actually 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*, possibly 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* label to the topic.

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.

## 5.2 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. Creating annotated topics (i.e., manual labels) for training is painstaking, but with a suitably representative set of topics, the effort is acceptable. To annotate *all* topics took us approximately 20 hours per project, but we estimate only 10% of the topics need annotation to produce useful results.

Very rarely did exp2 and exp3 (semi-unsupervised 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.

The semi-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, since the NFRs are typically loosely defined concepts in common parlance.

We found that the multi-label learners of BR, CLR and HOMER only did as well or worse for Macro-ROC as the single-label Naive Bayes and other naive Bayes-derived learners. This suggests that by combining together multiple Naive Bayes learners we could probably label sets of topics effectively, but it would require a separate Naive Bayes learner per label.

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 semi-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 maintaining acceptable run-time performance.

Since this work focuses on labelling natural language commit log comments we feel it can be adapted to other natural language artifacts such as mailing-list discussions and bug reports, even though that was not evaluated in this paper. Bug reports might not exhibit the same behaviour as commits in terms of dominant topics.

## 5.3 Threats to Validity

*Construct validity* – we used only commit messages rather than mail or bug tracker messages. To extend further we would need matching repositories for each project. Possibly they would have influenced our results, but there would be a degree of correlation between the corpora. Our taxonomy for software NFRs is subject to dispute, but seems generally accepted. Finally, there are exogenous sources, such as in-person discussions, which we did not access, an omission as noted in Aranda et al. [1]

*Internal validity* – 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. We were unable to assess inter-rater reliability.

*External validity* – Our data originated from OSS database projects and thus might not be applicable to commercially developed software or other domains. Furthermore, our analysis techniques rely on a project's use of meaningful commit messages, although we feel this is the most frequently occurring form of developer comments.

*Reliability* – each annotator, the first two authors, followed the same protocol and used the same annotations. However, only two annotators were used; their annotations could be biased as we did not analyze for inter-rater reliability because they did not rate the same documents.

## 5.4 Future Work

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.

# 6. CONCLUSIONS

This paper presented a cross-project data mining technique, *labelled topic extraction*. Previous topic analysis research produced project-specific topics that needed to be manually labelled. To improve on this, we leveraged software engineering standards, specifically the ISO9126 quality taxonomy, to produce a method of partially-automated (supervised) and fully-automated (semi-unsupervised) topic labelling. Since the word-list technique is not project-specific, we used it to compare two distinct projects, where we showed our technique produced interesting insight into maintenance activity.

We validated our topic labelling techniques using multiple experiments. We first conducted semi-unsupervised labelling using word-lists. Our next approach was supervised, using single-label and multi-label learners. Both kinds of learners performed well with average ROC values between 0.6 and 0.8. These results, along with the efficient performance of our learners, demonstrate that labelled topic extraction is a promising approach for understanding the occurrence of non-functional requirements in software projects.

# 7. REFERENCES

[1] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *International Conference on Software Engineering*, pages 298–308. IEEE, Sep 2009.

[2] 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.

[3] 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.

[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] 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.

[6] 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.

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

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

[9] S. Few. *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly Media, 1 edition, Jan. 2006.

[10] 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.

[11] 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.

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

[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] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

# APPENDIX

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