Empirical Software Engineering manuscript No.
(will be inserted by the editor)

Automated Topic Naming

Supporting Cross-project Analysis of Software Maintenance Activities

Abram Hindle · Neil A. Ernst · Michael W. Godfrey · John Mylopoulos

Received: date / Accepted: date

Abstract Software repositories provide a deluge of software artifacts to analyze. Researchers have attempted to summarize, categorize, and relate these artifacts by using semi-unsupervised machine-learning algorithms, such as Latent Dirichlet Allocation (LDA), used for concept and topic analysis to suggest candidate word-lists or topics that describe and relate software artifacts. However, these word-lists and topics 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 LDA from commit-log comments recovered from source control systems. These topics are given labels from a generalizable cross-project taxonomy, consisting of non-functional

Abram Hindle
Dept. of Computing Science
University of Alberta
Edmonton, AB, CANADA
E-mail: abram@softwareprocess.es

Neil A. Ernst
Dept. of Computer Science
University of British Columbia
Vancouver, BC, CANADA
E-mail: nernst@cs.ubc.ca

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

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

requirements. Our approach was evaluated with experiments and case studies on three large-scale Relational Database Management System (RDBMS) projects: MySQL, PostgreSQL and MaxDB. The case studies show that labelled topic extraction can produce appropriate, context-sensitive labels that are relevant to these projects, and provide fresh insight into their evolving software development activities.

Keywords Software maintenance · Repository mining · Latent Dirichlet allocation · Topic models

1 Introduction

A key problem for practicing software maintainers is gaining an understanding of *why* a system has evolved the way it has [26]. This is different from *how* a system has evolved. Looking back on streams of artifacts scattered across different repositories, inferring what activities were performed, when, and for what reasons, is hard without expert advice from the developers involved. In this work we provide a method of automatically labelling development topics extracted from commit logs, this method is called *labelled topic extraction*.

Concrete applications of *labelled topic extraction* include the annotation of development artifacts with NFR-related tags and the creation of project dashboards. Annotating software development artifacts with NFR-related tags would allow developers to create detailed directed queries of different artifact kinds that concern the same NFR; for example, a developer could browse the recent history of performance-related bug reports and code check-ins. Project dashboards distill detailed information about a software system into a simpler and more abstract view that summarizes key aspects of the development effort [10]; labelled topic extraction would allow managers to track effort related to specific NFR topics, such as *usability* or *portability*.

Topic modeling (such as Latent Dirichlet Allocation [2]) is a machine learning technique that 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 words that commonly occur together. Among other uses, topic modelling can quickly give developers an overview of where significant activity has occurred, and provide managers or maintainers an enhanced understanding of the project's 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 that automatically suggests project-independent labels for topics.

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) [18], 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 [19], 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.

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:

- We introduce the concept of labelled topic extraction, using a taxonomy of non-functional requirements (NFR) for our labels;
- We evaluate 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);
- We examine how NFRs correlate with the work of individual developers;
- We provide a method of cross-project analysis via topic labelling, and we apply 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 three 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.

This work extends our previous work [15]. The major extensions in this paper are as follows:

– We conducted another case-study on PostgreSQL.
– We used two authors to each annotate the same PostgreSQL topics in order to compare these annotations.
– We used this new case-study to test *inter-rater reliability* (IRR), described in Section 5.2.
– We conducted an analysis of PostgreSQL authors and their association with NFRs and topics (Section 4.3).

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) (software qualities) [6]. One approach they tried was similar to this one, with keywords mined from NFR catalogues found in Chung et al. [5]. Their approach resulted in 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 is different because we use a more comprehensive set of terms based on a taxonomy that is an integral part of our framework. Another difference is that we make cross-project comparisons instead of focusing on a single project. They relied on relatively well-structured requirements documents instead of version control histories that we use. The objective of Cleland-Huang's study was to identify new NFRs for system development, while our objective was to recover those latent NFRs from commit-log messages of the project.

Similarly, Mockus and Votta [26] studied a large-scale industrial change-tracking system. Mockus and Votta leveraged WordNet [9], 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 were not feasible (particularly in the distributed world of open-source software).

Another approach is to extract concerns from software repositories. Marcus et al. [22] used 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. [28] showed tag occurrence using colour and intensity, and our plots that rely on color and intensity have a similar look and feel to Treude et al.'s plots. They mined developer created change request tags from IBM Jazz repositories and used these to analyze the evolution of a single product. Change requests in Jazz allow for users to annotate each change request and check-in with tags such as "ui", "usability" or "milestone 3". The presence of a well-maintained set of tags is obviously essential to the success of this technique.

In Baldi et al. [1], 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. For example we might have a label called *"Oracle"* in the MySQL case, since Oracle owns MySQL. Our approach differs in two important ways: we automate the process of naming the topics, and we label topics with project-independent terms, in order to permit cross-project comparison.

Mei et al. [25] 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.

Massey [23] 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 [13] looked at GNOME specifically, and listed several sources for requirements: leader interest, mimicry, brainstorming, and prototypes. None of this work addressed quality requirements in OSS, nor did it examine requirements trends. In Hindle et al. [17] we examined release patterns in OSS. That work showed that there is a difference between projects regarding maintenance techniques. This supports the result described in this paper, that software qualities are not discussed with the same frequency across projects.

This paper and our MSR 2011 paper [15] are based off of the work of Ernst and Mylopoulos [7] and the work of Hindle et al. [18]. In Ernst and Mylopoulos [7], we describe an earlier project that identifies changes in quality requirements in GNOME software projects (GNOME is a Linux desktop environment). Unlike this paper, this earlier approach was more exploratory and had less validation. In particular, it uses different word-lists, solely uses text-matching, and does not leverage machine learning strategies. Our approach in this paper, and our MSR 2011 paper [15], extends Ernst and Mylopoulos [7] by using word-lists to label topics, which are completely absent in the earlier work. Hindle et al. [18] propose a windowed method of topic analysis that we extend with labelled topics, NFRs and new visualizations. This windowed method was to bucket documents by time windows (such as months), that

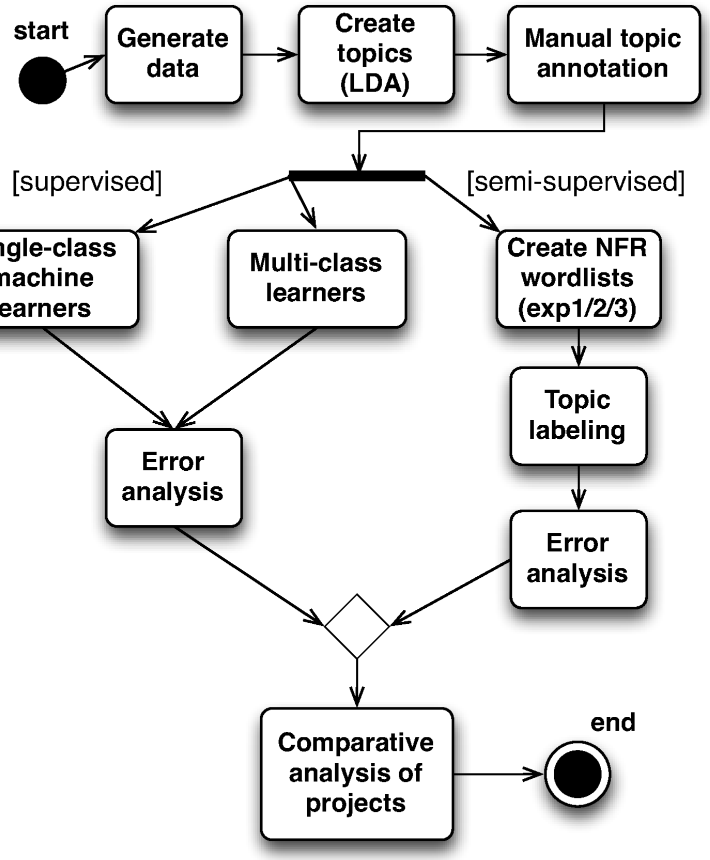


Fig. 1 Research methodology process view.

could overlap if necessary, and then use LDA to extract documents from these bucket. These topics were then related to topics in other adjacent buckets, and joined if deemed sufficiently similar. This paper, and our MSR'2011 paper [15], extend the method of Hindle et al. [18] by labelling topics and providing an alternative visualization, the topic time-line described in Section 4.

3 Study Design and Execution

Figure 1 gives an outline of our methodology. We begin by gathering source data and creating topic models. For semi-unsupervised labelling, we generate three sets of word-lists as signifiers for NFRs. With supervised learning, we train our data with manual annotations in order to match topics with NFRs. Finally, these topics are 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 used three different open-source, partially-commercial database systems:

MySQL 3.23 — Started in 1994 and MySQL 3.23 was released in early 2001. MySQL contains 320,000 lines of C and C++ source code [1]. We used the MySQL 3.23 source control history from July 31st, 2000 to August 9th, 2004.

MaxDB 7.500 — 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 [2]. We used the MaxDB 7.500 source control history from June 29th, 2004 to June 19th, 2006.

PostgreSQL 7.3 — Started in the 1980s as a Berkeley research project [3]. PostgreSQL 7.3 contains 306,000 lines of C code. We used the PostgreSQL 7.3 source control history from May 9th, 2002 to August 26th, 2004.

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. We felt that a single domain would allow for cross-project comparison. At the same time we recognize that problem domain alone does not guarantee functional and non-functional similarity. For instance, each database system has a different focus. PostgreSQL tends to focus on fulfilling much of the SQL92 specification and adding more features, while MySQL has been slow to adopt much of that specification in favour of community-requested features. The consequence of our choice to look at a single domain is to limit generalizability. What we show in this paper will be somewhat biased towards database software.

For each project, we studied source control commit comments, the messages that programmers write when they commit revisions to a source control repository. Most commits we observed had comments: 90% in MySQL 3.23, 98.5% in PostgreSQL and 99.99% in MaxDB 7.500. 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., CVS). 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.

[1] generated using David A. Wheeler's *SLOCCount*, http://dwheeler.com/sloccount.
[2] http://www.sdn.sap.com/irj/sdn/maxdb
[3] http://www.postgresql.org/docs/7.3/static/

Each word in the message was stripped of punctuation and converted to lowercase. We summarized each message as a word distribution minus stop-words such as "*the*" and "*at*". We did not stem or apply any other transforms to the messages. Our stop words are derived from the Natural Language Toolkit (NLTK) English stop-word list [4].

For the commit message data-sets of each project, we created an XML file that partitioned 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 [18]. 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 extracted.

A topic analysis tool such as LDA will try to find N independent word distributions within the word distributions of all input messages. If there are not N independent word distributions, the topics produced tend to be duplicates of each other, that is, they share the top terms. During this study we found that if we used a value of 20 for N, duplicate topics were infrequent. Linear combinations of these N word distributions are meant to represent and recreate the word distributions of any of the original messages. In other words, these topics are 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.

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 [19], which describes six high-level NFRs: *maintainability, functionality, portability, efficiency, usability,* and *reliability.* There is some debate about the terms in this model [3], and whether they are a) the correct terms and b) correctly organized. However, ISO9126 is "an international standard and thus provides an internationally accepted terminology for software quality [3, p. 58]," and so we consider that it is sufficient for the purposes of this research. *Performance* is an example of an RDBMS word related to the *efficiency* NFR. We claim that these NFRs are maintenance concerns (to varying degrees) in all software projects, and are therefore well suited for comparisons between projects.

3.1.2 Creating a Validation Corpus

To evaluate both semi-unsupervised and supervised classification, we created a validation set of manually labelled topics. Per each project, the annotators (the first two authors) annotated each extracted topic in each period with the

[4] NLTK: `http://www.nltk.org/`

six NFR labels listed above. Annotators did not annotate each other's annotations, except that PostgreSQL (PgSQL) was annotated by both annotators in order to evaluate inter-rater reliability. We looked at each period's topics, and assessed what the data—consisting of the frequency-weighted word lists and messages—suggested was the most appropriate labels for that topic. We selected the appropriate labels 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 *portability*, *efficiency*, *reliability*, *functionality*, *usability*, and *maintainability*, the annotators annotated each topic with the relevant label. We added a catch-all label, *none*, which we used when none of the six NFRs was suitable. In some instances, we used finer-grained annotations that would be aggregated up to one of these higher-level labels.

We validate classification performance using the *area under the curve of Receiver Operating Characteristic* [8], abbreviated *ROC*, sometimes called Area Under the Curve or AUC, and the F-measure, which is the harmonic mean of precision and recall, i.e., $2*(P*R)/(P+R)$. Throughout the paper we will provide F-measure scores so that readers who are more familiar with F-measure than ROC can intuitively interpret the results.

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). While we recognize that using 0.5 as the base-line means our ROC scores will look much larger than our F-Measure scores, we feel that the knowledge that random selection is 0.5 or worse is helpful for interpreting our results. 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 argue for using ROC over F-Measure because ROC suffers less from bias, as F-Measure often skews towards the positive class [11,12], especially in the case of class imbalance. Forman et al. [12] demonstrate that cross-fold validation will often produce low F-measures given high class imbalance and the presence of false negatives. Although recent work [12] has suggested that while ROC suffers less bias than the average F-measure (F_{avg}) for cross-folds, F-measures ($F_{tp,fp}$) that are calculated from the sums of true-positives, false-positive, and false-negatives across folds tend to exhibit no bias. Unfortunately our experimental framework lacks a robust way to calculate $F_{tp,fp}$, thus we still provide ROC and F_{avg}. 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. We call this “semi-unsupervised” because while there is no “training set”, we do seed the word-lists manually.

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 or *word-lists*, in our parlance. These word-lists were determined a priori and were not extracted from the projects themselves, using the methodology explained below. In general, these lists are project and domain independent. 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 for software quality measurement described in Kayed et al. [20], which was constructed using 80 source documents, including research papers and international standards. The labels we used were:

> 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 recognize that terms like this might be used in a different context with a different meaning, like code-cloning. We therefore took the NFRs from the ISO9126 and added terms to them.

To construct these expanded word-lists, we used WordNet [9]. 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. [24]. We then conducted a simple random analysis of mailing list messages from an open source ecosystem, KDE. Like the GNOME study we conducted in Ernst and Mylopoulos [7], KDE contains a suite of different products covering a variety of software categories. 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. Table 1 shows the labels (NFRs) and word-lists we used for matching.

For the third set of word-lists, exp3, we extended the word-lists from exp2 using WordNet similarity matches. Similarity in WordNet means siblings in a

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`

hypernym tree. We do not include these words here for space considerations [5]. Wordnet similarity is a very broad match. For example, the label *maintainability* is associated with words *ease* and *ownership*, and the word *performance* has a 'sense' that refers to musical performances, which is obviously unrelated to software development. In general, as we proceed from word-lists 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 frequency distribution of words that constitute the topic. A *named topic* is a topic with a match. *Unnamed topics* occur where there is no such match. This may indicate either a lack of precision in the word-lists, or simply that this topic is not associated with non-functional requirements. All experiments were run on the data-sets for each project (e.g., PostgreSQL, MySQL, MaxDB). LDA extracted 20 topics per period for each project. Each change-log message was lightly processed before applying LDA: words were converted to lowercase with punctuation removed and then stop words were removed. This labelling is *semi-unsupervised* because the corpus is not derived

[5] For our word lists visit `http://softwareprocess.es/nomen/`

Project	Measure	exp1	exp2	exp3
MaxDB 7.500	Named Topics	305	183	330
	Unnamed Topics	84	206	59
MySQL 3.23	Named Topics	341	202	469
	Unnamed Topics	245	384	117
PgSQL 7.3	Named Topics	639	543	640
	Unnamed Topics	1	97	0

Table 2 Automatic topic labelling for MaxDB, MySQL and PostgreSQL

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, we can use the domain knowledge of software to label relevant topics.

Table 2 shows how many topics were labelled for MaxDB, MySQL and PostgreSQL. Notice how PostgreSQL had far fewer unlabelled topics, PostgreSQL also was under going far more development as we evaluated 7744 PostgreSQL commits versus 8664 MaxDB commits and 6223 MySQL commits. In terms of change-log comment words PostgreSQL had 164724, while MaxDB had 68203, and MySQL had 101081. This implies there were more PostgreSQL terms for the topic analyzer to produce topics with. And thus we suspect that PostgreSQL topics were flooded with many terms, indicating that perhaps we needed more than 20 topics per month for PostgreSQL, as this result indicates there is a lot of overlap in the topics.

For `exp1` the labels with the most topics were *correctness* (182/305/640, which represent MySQL, MaxDB and PostgreSQL topic counts, respectively) and *testability* (121/238/625). We did not see many results for *usability* (4/0/138) or *accuracy* (3/0/27), which were infrequently matched. Note the significantly higher result for *usability* in PostgreSQL—this suggests a difference in how project developers are discussing usability, at least with respect to our analysis. We also looked for correlations between our labels: excluding double matches (self-correlation), our highest co-occurring labels were *verifiability* or *correctness* with *traceability*, and *testability* with *correctness*.

For `exp2`, there are more unnamed topics than `exp1`. Only *reliability* produces the most matches, mostly due to the word "error". Co-occurrence results were poor. This suggests our word lists were overly restrictive. For PostgreSQL *reliability* and *usability* co-occurred with *portability* and *efficiency*.

For `exp3`, we generally labelled more topics. As mentioned above, the word-lists are broad, so there are likely to be false-positives (discussed below). The most frequent label across all projects (for this word-list, and unlike `exp1`) was *usability*, and the least frequent label was *maintainability*. This implies that our signifiers for usability in this experiment were fairly broad. Common co-occurrences were *reliability* with *usability*, *efficiency* with *reliability*, and *efficiency* with *usability*.

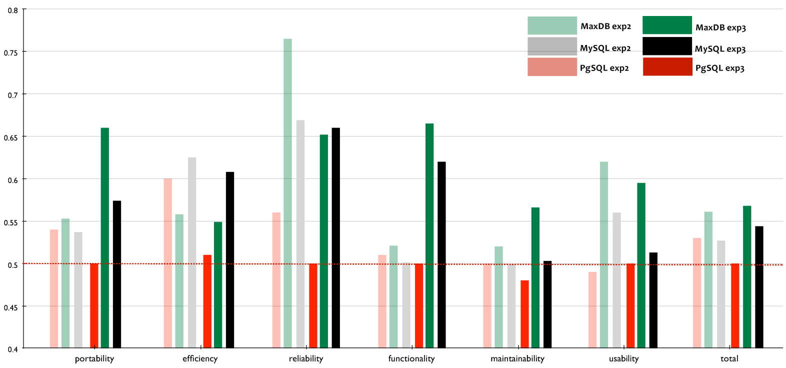


Fig. 2 Performance, ROC values (range: 0–1), of semi-unsupervised topic labelling for each NFR and per word-list. The dashed line indicates the performance of a random classifier. This graph shows how well the semi-unsupervised topic labelling matched our manual annotations.

3.2.3 Analysis of the Semi-unsupervised Labelling

For each quality in the high-level ISO 9126 taxonomy (namely, *Maintainability, Usability, Reliability, Efficiency, Portability, Functionality*) we assessed whether the semi-unsupervised labels for a topic matched the manual annotations we created for the validation corpus. Recall that the manual annotations were not used to train the labelling process. As described in Section 3.1 we used both ROC and F-measure measures to evaluate the performance of the classification. Figure 2 shows our ROC results for PostgreSQL, MaxDB and MySQL. We omit plots of `exp1` due to poor results. We describe F-measure results in the text below.

Because our ground truth annotations were relevant only to ISO9126, `exp1` had poor performance due to the overlap between ISO9126 and the Kayed ontology (i.e., we annotated topics with labels which did not appear in the validation corpus). For `exp1` the F-measures for the NFRs for MaxDB were from 0 to 0.18 with an average (of all NFRs) of 0.03, for MySQL were from 0 to 0.16 with an average of 0.05, and for PostgreSQL 0 to 0.15 with an average of 0.07.

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. PostgreSQL had an average F-measure of 0.30 with a range of 0.09 to 0.38. MaxDB had an average precision and recall of 0.25 and 0.22 while MySQL had 0.41 and 0.10 and PostgreSQL 0.31 and 0.29, 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. PostgreSQL had an average F-measure of 0.42 with a range of 0.31 to 0.54. MaxDB had an average precision and recall of 0.16 and 0.67 while MySQL had 0.3 and

0.48. PostgreSQL had average precision and recall of 0.27 and 0.95, respectively.

Based on this we found 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. For PostgreSQL in `exp2`, *reliability* and *efficiency* were likewise the most accurate, while *functionality* remained poor. *Functionality* improved dramatically by `exp3`. Our F-measure scores were low and many ROC scores were 0.6 or less, but our classifier, in most cases, still performed substantially better than random (0.5), even in the face of heavy class-imbalance for qualities such as *usability* and *efficiency*. While there is much room for improvement, we are seeing some correlation between our quality word lists and relevant topics.

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 [19] for our classes, but other taxonomies are also applicable.

We used a suite of supervised classifiers, WEKA [14], that includes machine learning tools such as support vector machines and Bayes-nets. We also used the multi-labelling add-on for WEKA, Mulan [29]. 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. For example, a given topic (word distribution) may be 'about' both usability and maintainability, if this topic was a product of a discussion on design trade offs. 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 [21], a common technique for evaluating machine learners, where the original data is partitioned randomly into ten sub-samples, and each sample is used to test against a training set composed of the nine other samples. We discuss these results below.

3.3.1 Analysis of the Supervised Labelling

Because our data-set consisted of word counts we expected Bayesian techniques to perform well. Bayesian techniques are often used in spam filtering, which is similarly interested in word distributions. We tried other learners that WEKA [14] provides, including 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, per each NFR one should

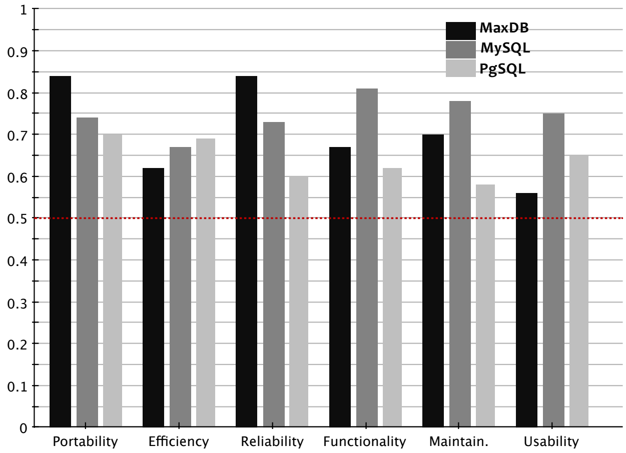


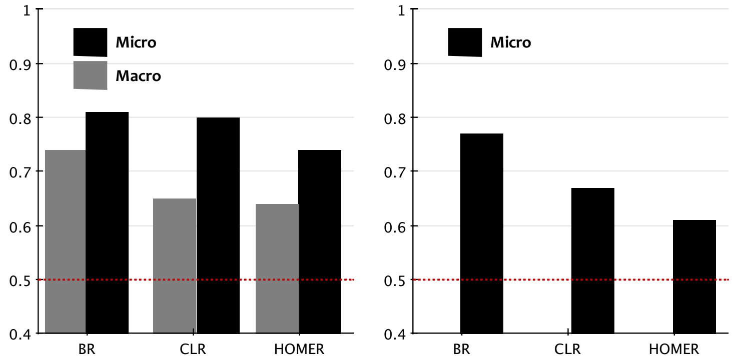
Fig. 3 ROC value for the best learner per label for MaxDB, MySQL and PgSQL. Values range from 0–1. Dashed line indicates the performance of a random classifier.

select the best performing learner possible. Given the tractability of these techniques, a tool which applied all learners and presented the best result should be feasible.

Figure 3 shows that MaxDB and MySQL have quite different results, as the ROC values for *reliability* and *functionality* swap between projects. For PostgreSQL, the performance is nearly always poorer than the other two systems. The reason for this lack of performance could be that parameter we chose for number of topics, N, could be non-optimal for PostgreSQL. One explanation is that given the size of the PostgreSQL datasets, it was becoming hard to distinguish one topic from the next. PostgreSQL's data-set was the largest of the 3 projects: the XML datable that described the PostgreSQL topics that we annotated was 8 times larger than MySQL and 1.7 times larger than MaxDB. These size differences arise from the number of commits, the number of files and the verbosity of the commit descriptions.

Although we used a variety of machine learners, we found that Bayesian techniques performed the best on all projects. We believe this is due to the large number of features they can handle. Our best-performing learners—Discriminative Multinomial Naive Bayes, Naive Bayes and Multinomial Naive Bayes—are all based on Bayes' theorem and all make the naive assumption 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 training on or classifying one instance with 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 a mean of 0.48. MaxDB had a range of 0.17 to 0.61 with a mean of 0.39. Finally, PostgreSQL had a range of 0.04 to 0.9 and a mean of 0.43.



(a) MySQL (b) MaxDB

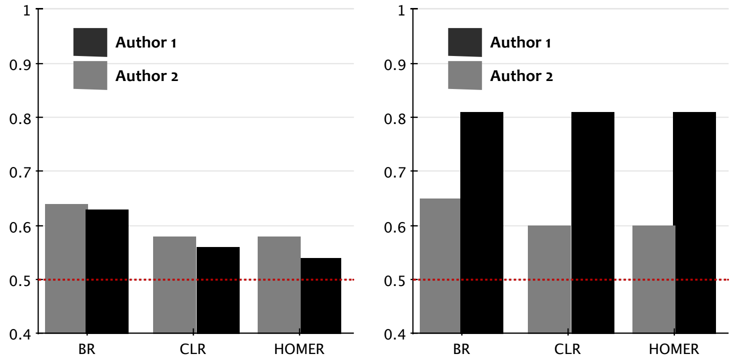
Fig. 4 MySQL and MaxDB macro and micro-ROC results per multi-label learner. Possible values range from 0–1. Dashed line indicates the performance of a random classifier.

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*, MySQL *efficiency*, and PostgreSQL *maintainability*). 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 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 relates to *usability*. However, traditional machine learning techniques, such as Naive Bayes, can map topics to only a single class. The Mulan [29] library encapsulates several different multi-label machine learners which can label elements with multiple labels. Mulan also includes methods for determining the performance of these learners.

Two perspectives used to evaluate multi-label learners are with micro or macro measurements (shown in Figure 4a). 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.During cross-folds validation if the class imbalance or poor learner performance causes a division by zero when evaluating one of the NFR labels (in this case *usability*), cre-



(a) Macro-ROC (b) Micro-ROC

Fig. 5 Author #1 and author #2 ROC results for PostgreSQL.

ating an undefined value. This undefined value is propagated through to the mean of ROC per NFR label, causing the end calculation of macro-ROC to be undefined.

Figure 4 presents the results of Mulan's best multi-label learners for the MaxDB and MySQL projects, and Figure 5 for PostgreSQL. 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 [29].

Figure 5 shows the ROC results for the PostgreSQL product. In this figure, we show the relative differences when we use different training data-sets from different annotators. In 5a we see that ROC results are very similar. In the other figure, however, author #1 has dramatically better performance. We speculate that this is due to the particular annotation decisions made by author #1; in some sense he performed better. The difference in Macro-ROC was not significant, but the difference in Micro-ROC was, as the p-value of Student's T-test was < 0.001.

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. In this section we demonstrate the value of labelled topic extraction in addressing this issue.

Labelled topics address *why* questions by associating a commit with the underlying software qualities which motivated that change. The *how* of a change is the change itself, the software quality behind it, the *why* is what we are after. We investigate the history of the three 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 three research questions:

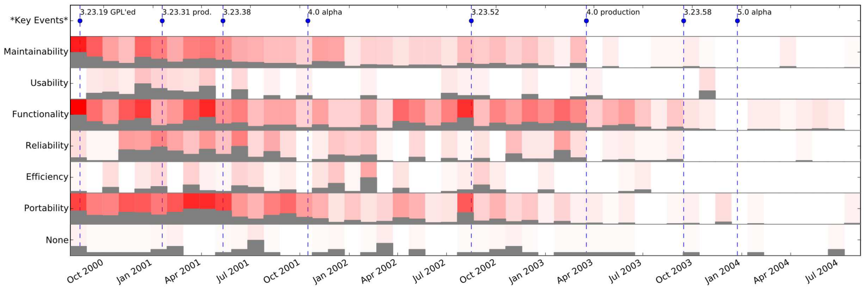
RQ1. *Do NFR frequencies change over time?* If a particular NFR was of more interest at one point in the life-cycle than another, this suggests that development activity shifted focus. For example, if a developer expected to see a recent focus on *reliability*, but instead *usability* dominated, they might re-prioritize upcoming work items.

RQ2. *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 question could be used 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 projects.

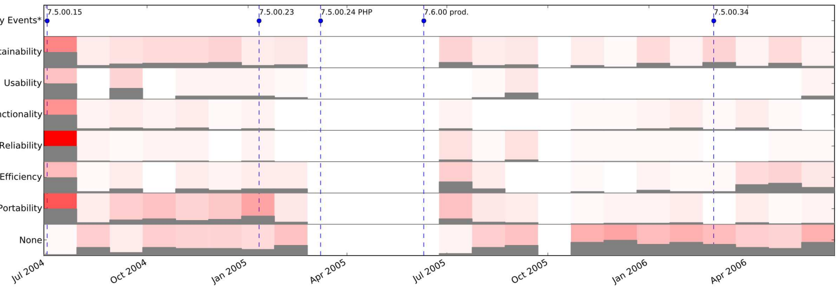
RQ3. *Do different developers work on different NFRs?* For a given project, it is reasonable to think that developers are either assigned (in commercial organizations) or choose (in open-source organizations) to work on a particular NFR. For example, one developer might be more junior, and take responsibility for the low-impact reliability fixes. Another, more senior developer might assume responsibility for major improvements such as efficiency improvements.

Topic time-lines are depicted in Figures 6a, 6b and 6c. These topic time-lines show the temporal patterns of NFR frequencies. This is generated from the manually annotated topics, although this visualization can be generated from the results of labelled topic extraction. Note that there are no unlabelled topics in this data-set.

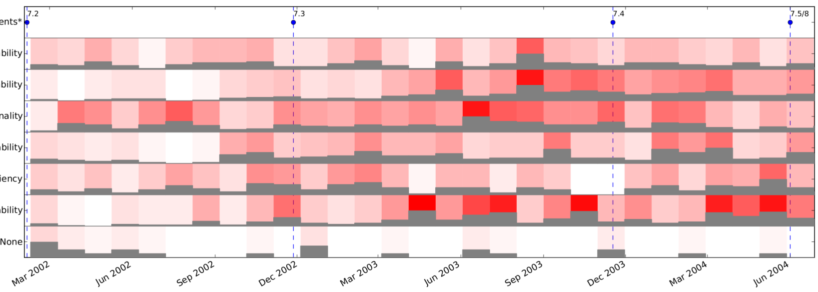
There are two measures represented. One, the relative frequency, shown in the grey histogram boxes, represents the number of topics with that NFR in that period, relative to the maximum number of topics assigned to the NFR. For example, in Figure 6a we see a spike in *portability* and *functionality* frequency in September 2002. The second, absolute frequency, is shown using cell intensity, and compares the number of topics labelled with the NFR per period relative to the maximum number of labelled topics overall. For instance, Figure 6a 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* in PostgreSQL, which shows periodic activity, and suggests that efficiency-related changes have longer lasting effects. A more systematic analysis of periodicity is necessary to properly conclude this. The topmost row in each diagram lists historical events for that project (such as a release).



(a) MySQL 3.23



(b) MaxDB 7.500



(c) PostgreSQL 7.2

Fig. 6 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.

We analyzed each project's developer mailing list for external validation (the body of the email). We use *labelled topic extraction* to pick out the underlying NFR activity behind these events. For example, both MaxDB and MySQL 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 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 6b) 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 development period of MySQL we studied (Figure 6a) saw the first releases to be licensed under the GPL. Version 3.23.31 (January, 2001) was the production release (non-beta), and the time-line view shows 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) and subsequently becomes more functionality oriented.

By contrast, the PostgreSQL time-line (Figure 6c) is extracted from a central trunk and is not version-specific (due to differences in how the projects manage branches). Therefore development tends to be focused on releases 7.3, 7.4 and 7.5/8.0 alpha. For example, a priority for the 8.0 candidate was a Windows-native port of the source code, which seems to correlate with the *portability* NFR increasing in frequency in mid-2004. Before the 7.4 release, *usability, functionality* and *portability* all increase in frequency, possibly reflecting the interest in adding features and documentation for the release. In the following sections we turn to our research questions:

4.1 RQ1: Do NFR Frequencies Change Over Time?

In both MaxDB and MySQL 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.

In PostgreSQL, by comparison, the frequencies seem to become somewhat cyclic, since we are not studying a maintenance-phase for the product, but rather ongoing feature addition and usability improvements.

4.2 RQ2: Do Projects Differ in Their Relative Topic Interest?

We found significant overall variation in the projects. 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). PostgreSQL had more NFR topics in general, but *portability*, *functionality* and *usability* were more prevalent in PostgreSQL. One notable difference between PostgreSQL and the other two projects is that many of the commits for PostgreSQL were UI changes and documentation changes. PostgreSQL also seemed to focus more on *reliability* later in the case study than earlier like MySQL. In all 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.

4.3 RQ3: Do Different Developers Work on Different NFRs?

We wanted to see whether developers in the PostgreSQL project[6] worked on similar NFRs. For example, one developer might be a usability expert, while another developer is focused on security. We assume that the NFR labels associated with a commit implicitly reflect work on those NFRs. Being able to report on where effort is being expended is useful for tracking productivity (among other things). We explored this issue by extracting, for each topic in the PostgreSQL data-set, the developer names responsible for the commits used to generate that topic, and the NFRs that are associated with the topic. The analysis in this section used Neil's annotations. This produced a map which associated developer name with NFR labels. The six NFR labels (*maintainability, usability, efficiency, functionality, portability, reliability*) and the *None* label describe a 7-dimensional space (essentially a distribution), into which we can position developers, based on the frequency with which each label occurs in the map.

Using this space, we first conducted pairwise X^2 (chi-squared) tests and Kolmogorov-Smirnov tests for each developer pair on their NFR distributions (18 developers, 306 pair-wise tests). We found that in 27% of these tests, the two developers were significantly different (i.e. distant in the 7-d space and have significant p-values less than 0.05). This implies that there are developers who focus on different NFRs or different proportions of NFRs, but the majority of developers have similar NFR distributions.

In Figure 7 we clustered the developers from PostgreSQL using the Ward method of Hierarchical clustering on the 7-dimensional space. This will group developers according to the Euclidean distance metric. The height in Figure 7 describes the euclidean distance between elements, in terms of the path We

[6] Since the MySQL and MaxDB data had poor records for developer ids, we focused on PostgreSQL.

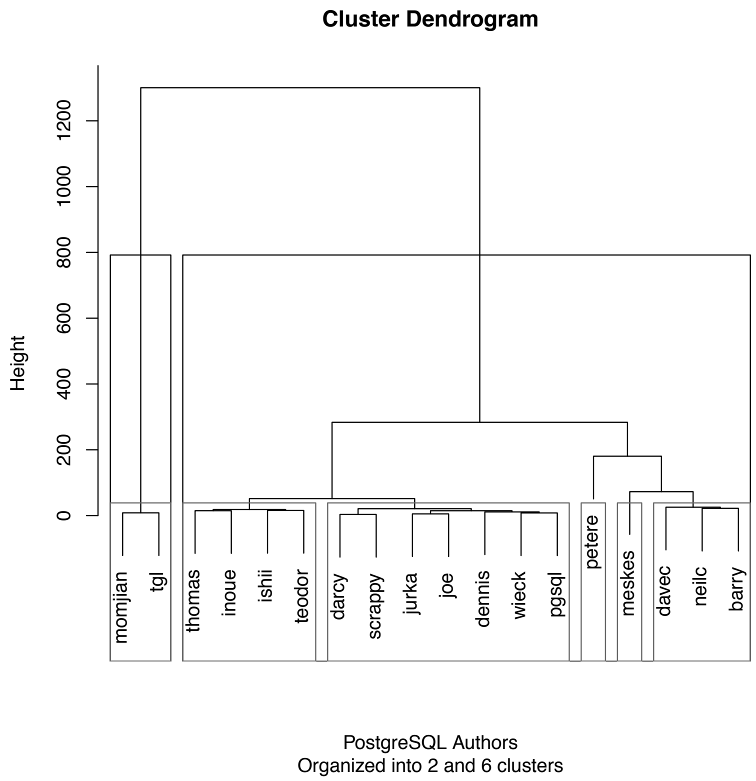


Fig. 7 PostgreSQL Author Clusters: Authors clustered by Euclidean distance of their NFR contributions. The outer rectangles (black) reflect the clustering of authors into 2 clusters; while the inner rectangles (dim grey) reflect clustering of authors into 6 clusters. The height is a measure of cluster similarity, Euclidean distance, where smaller values are more similar. Individuals that share branches are more similar to each other individuals who do not. The longer the path, in terms of height, between two individuals, the more different their NFR contributions are.

clustered the authors into both 2-clusters and 6-clusters. We chose 6-clusters because the difference in distance between entities and centroids was minimal at 6-clusters, and the derivative of distance was the highest in that region as well.

Figure 7 shows that when we use 6-clusters, `petere`, one of the major contributors, is in his own cluster while two other major contributors, `tgl` and `momjian` form their own cluster (i.e., they are distant from one another in the 7-dimensional space). This is interesting because it means that the important developers are have different proportions of NFR-relevant contributions; they have different focuses during development. In terms of the 2-clusters, we can see that `momjian` and `tgl` are in the same cluster, but `petere` is not. The most frequent committers do not share the same clusters, even at the coarsest level

NFR	Developer
Usability	`dennis, neilc`
Portability	`scrappy, meskes`
Efficiency	`inoue, neilc`
Reliability	`jurka, joe`
Functionality	`thomas, weick`
Maintainability	`scrappy, ishii`

Table 3 Developer interest in NFRs

of clustering, 2 clusters. This implies that developers in this sample do work on different sets of NFRs and have different software quality focuses.

If we compare the global NFR distribution (that is, the relative global frequency of each NFR label) to each author we find that 25% of the authors have a similarity (Pearson correlation) of 0.47 or less. In other words, for these authors (gathered to the lower left of Figure 8), their NFR distribution does not match well with the global distribution. Authors `momjian` and `tgl` are in the top right of Figure 8, dominate in topic count and commit count, and so are very similar to the global distribution of NFR topics. Figure 8 demonstrates that although a few authors dominate in the number of commits, many authors exhibit different behaviour in terms of the NFR-relevant topics of their commits. Furthermore, there are no authors in the top left corner, indicating that the authors who commit less, do not contribute with a similar NFR-topic distribution as `momjian` or `tgl`. We found that number of commits correlated with an author's similarity to the global NFR distribution (0.59 Pearson), i.e., the variables "number of topics associated with a developer" and "number of commits" are not discriminative.

Our working theory is that less frequent committers are more specialized, e.g., interested in a specific NFR, while the main developers (i.e., frequent committers) either have wider responsibility or have more time to be widely involved. An interesting extension would be to compare this data with the PostgreSQL source code files, to see which developer touched which file.

We say that an author (read developer) is "proportionately interested" in an NFR if, for all NFRs with which he or she is associated, a given NFR receives the plurality of his or her commits. This is a measure of relative interest and is independent of number of commits (subject to the caveat about frequent committers, above). If we look at which of the top 15 developers were proportionately interested in a given NFR, we find the associations identified in Table 3. Our data also showed that for these top developers, between 1/10 and 1/3 of their commits were labelled with a single NFR.

Based on Figure 8 we wanted to see which clusters `momjian`, `tgl`, and `petere` would be in if we evaluated the clusters proportionately, that is if we use distance measures that normalize the data and ignore magnitude (e.g. turning an author's NFR contributions into a histogram or unit-vector) such as cosine distance, Euclidean distance of unit vectors, or Pearson correlation distance ($1-r$). Whereas, Euclidean distance uses both angle and magnitude of

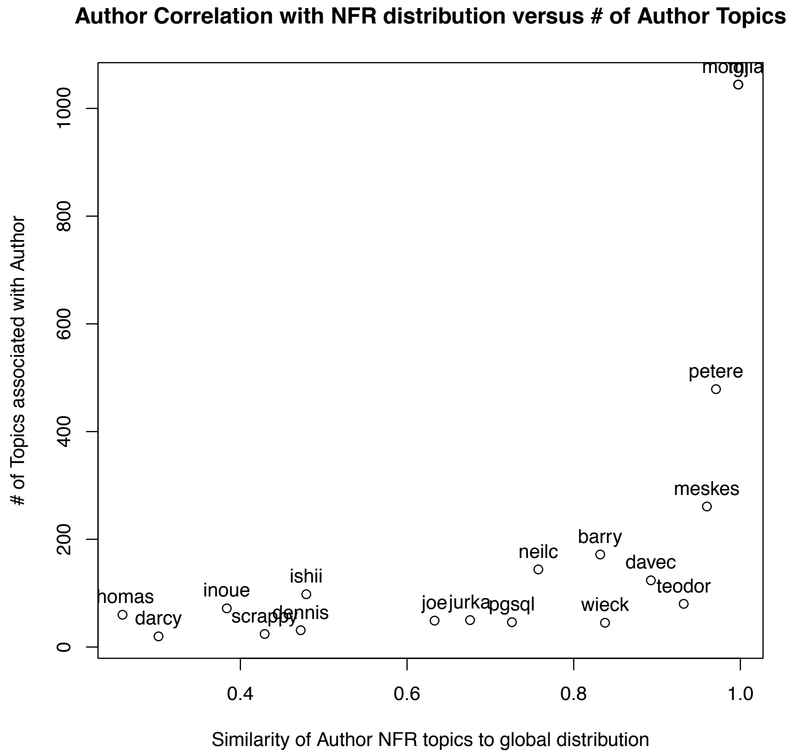


Fig. 8 Author commit count versus the similarity of the NFR distribution of an Author to the Global Distribution of NFR topics. Similarity is Pearson correlation between the counts of NFR relevant topics associated with an Author and the total counts of NFR relevant topics.

a vector. For all three non-magnitude distances measures we tend to see that the three largest contributors, `momjian`, `tgl`, and `petere` inhabit the same clusters when we use 2-clusters or 6-clusters. This confirms the observations in Figure 8, which uses Pearson correlation, that the larger contributors tend to be proportionally similar.

One potential confound for this analysis is that we could be describing developer style instead of developer focus. Developer style would be a developer's likelihood to use terms found within our dictionaries and training corpus in their commit messages. Furthermore, a top developer, in terms of number of commits, will have more samples, and thus be more likely take on a more general role. For example, we found in a previous project [16] that there was a good correlation between the words used to describe a commit and the author of the commit.

	Cohen's Kappa	Spearman Correlation
Portability	0.154	0.253
Functionality	-0.014	-0.014
Reliability	0.005	0.005
Maintainability	0.082	0.082
Efficiency	0.231	0.258
Usability	0.009	0.014
None	0.062	0.081
Everything	0.107	0.108

Table 4 Inter-rater Reliability on PostgreSQL

5 Discussion

5.1 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*, possibly because concurrent code is prone to bugs such as race conditions. Topics related to *configuration management* and *source control* 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 concerned 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: all three projects, MySQL, MaxDB, and PostgreSQL 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 Inter-rater reliability

To determine inter-rater reliability two of the authors—Ernst and Hindle—each annotated the PostgreSQL topics, and then evaluated each other's annotations. Table 4 describes the Cohen Kappa and the Spearman correlation of our per-topic annotations for each NFR. We evaluated inter-rater reliability using each NFR, because a single topic could be tagged with more than one NFR.

These results are fairly poor. The aggregate view of a Kappa of 0.1 indicates there is some weak agreement. We found that there was good agreement in terms of lack of an annotation, but disagreement regarding which annotation to apply. After some discussion we concluded that *usability* was a primary source of disagreement. For instance, should we annotate a commit which updates the user manual as a usability-related change? Is adding a command-line option a usability issue? These kinds of questions illustrate some of the agreement, disagreement, and ambiguity about these labels. We therefore recommend that future annotators train and discuss how and when an annotation is appropriate. This ought to be easier to do if an internal team is using our tool, since (presumably) there is a greater shared understanding of project activities, and therefore, potentially greater rater agreement.

To evaluate our results empirically, we compared them to the annotations that would result if the seven labels were applied randomly. In the first simulation, as shown by Figure 9, we sampled with replacement from our own NFR label distributions to produce random ratings that looked like our own. We then applied the Kappa-statistic on this sampling and our labelling, and repeated this 100,000 times. We then compared this distribution of simulated IRR ratings against our own IRR ratings. We found that for 4 out of 7 NFR labels, the IRR values for our labels were greater than the random IRR measures 96% of the time. For *portability* and *efficiency*, our IRR was greater 100% of the time both for Neil's ratings, Abram's ratings and the union of both Neil and Abram. We know this by evaluating our IRR ratings against the empirical cumulative distribution function (ECDF) of the IRR ratings of the 100,000 simulation runs. One explanation of the positive result for *portability* is that such changes are often accompanied by clear indicator terms such as *Windows, Linux, SunOS*, etc.

Figure 9 depicts the IRR of these simulated sampled ratings versus our IRR using the Kappa statistic. We also achieved similar results if the ratings were pulled from uniform distributions. These experiments indicate that our labels and ratings were an improvement on random annotations in all cases, with the exceptions of *functionality*, *reliability* and *usability*.

Were our results harmed by low IRR scores? For MaxDB and MySQL, since each annotator (Neil or Abram) acted as the sole oracle, the results for either system still stand, but construct validity is harmed since it is not clear if the annotated NFR actually represented the NFR relevant to that commit. The measured empirical performance of each system still stands, but the comparison between systems in terms of NFRs might be harmed by low IRR scores.

The low IRR scores indicate the underlying difficulty of annotating such a data-set. To improve IRR, consistent training of the annotators or the use of more annotators might help. Other improvements could be gained by approaching some of the original developers and using their expertise to annotate the data according to their original, recollected intent. Improvements could be therefore be gained by producing more robust training samples that use multiple raters, or experts, who are similarly trained and willing to discuss and negotiate disagreements.

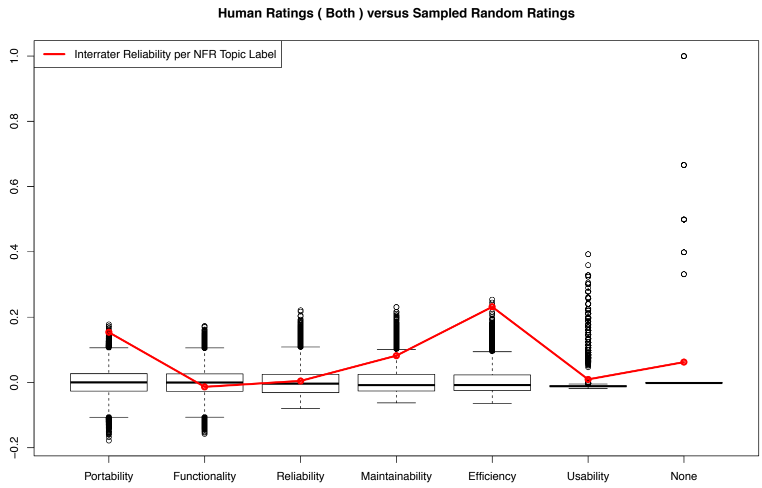


Fig. 9 Measured IRR versus IRR of random labeling simulations. Perfect IRR is 1.0. The red line and points highlight the measured inter-rater reliability of NFR topics labels between annotators of PostgreSQL. The box-plots are the distribution of 100,000 IRR ratings between random simulation, drawn from our distribution of ratings, and our labelling of NFRs. Note how for 4 of the 7 NFRs the measure IRR is distinctly higher than the median of the simulations.

5.3 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, we feel that the effort is acceptable for non-academic use. 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 MySQL and MaxDB, yet for PostgreSQL the `exp2` word-lists performed better. For both MySQL and MaxDB *usability* was better served by `exp2`. *Usability* was a very infrequent label, however, which made it difficult to detect for both approaches.

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 performed only 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 for MySQL and MaxDB and 0.47 to 0.6 for PostgreSQL, we can see there is promise in supervised 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. Low F-measures and ROC scores are a concern for some of these techniques, perhaps the word lists need to be re-enforced or made robust in the face of heavy class imbalance. These results are promising because they indicate that these techniques are accurate enough to be useful while still maintaining acceptable run-time performance.

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

5.4 Threats to Validity

Our work faced multiple threats to validity and we have attempted to address them:

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. It is possible for a given label to occur across the arbitrary 30-day boundary we set. We suspect but have not proved that this is insignificant. Our taxonomy for software NFRs is subject to dispute, but seems to be generally accepted. A future approach should consider a different taxonomy, such as one created by surveying developers on what "types" of tasks they work on. Finally, there are exogenous sources, such as in-person discussions, which we did not access.

Our word-lists were built up of words that were assumed to be relevant to those topics, our automated analysis was ignorant of the multi-uses of words and thus topics could be flagged inappropriately. For instance, if the word *redundancy* is used would it reference reliability or cloned code? This issue is why we checked the performance of the techniques, although we did not explicitly check for these cases.

Developer style is another confounding issue; if we are searching for words in a word-list, we are relying on developers to use these words. This study might be exploiting the behaviour or style of a few developers. If one developer did not describe their commits well or used fewer terms it is likely they would be associated with a NFR topic regardless of the actual purpose of their commits.

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.

PostgreSQL was larger than MySQL and MaxDB. This imbalance in size, combined with the choice of 20 topics per month produced topics that represented too many issues or threads within PostgreSQL's development. Choice of number of topics should probably be tuned to the project.

We primarily relied on ROC (the area under the receiver operating characteristic curve), as our measurement to compare the effectiveness of semi-unsupervised and supervised learners. ROC is less biased than F-Measure in cross-folds validation [11,12] in the case of class imbalance. We feel that the fixed point of 0.5 for random results allows us to better describe the predictability achieved by some of these techniques against random and ZeroR learners. We also include F-Measure results to allow for the reader to validate against F-Measure if they are more comfortable with it. Thus because of F-Measure's biased handling of class-imbalance and the inherent class-imbalance that our data suffers from we chose ROC.

One potential issue with some of our results is that we often achieved very low ROC and F-Measure scores. Performance was not uniformly low, as the more common classes often exhibited better performance than the more rare classes. Often this was amplified by class imbalance. In our future work we plan to investigate techniques such as sub-sampling and boot-strapping in order to improve performance. Some of the ROC performance could be due to a lack of coherence in tagging which was shown by our low inter-rater reliability score. Inter-rater reliability is a threat, and we discussed it in Section 5.2.

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. While we tried to control for application domain variability, OSS projects investigated were database systems, thus our results might not generalize well to other domains. Generalizability is enhanced by the fact that all three projects were maintained by different developers and different teams using different development processes.

Reliability – each annotator, the first two authors, followed the same protocol and used the same annotations. However, only two annotators were used; their annotations exhibit some bias as suggested by the weak inter-rater reliability for PostgreSQL. Inter-rater reliability could not be checked for MySQL and MaxDB because annotators did not rate the same documents. This is discussed at length in Section 5.2.

5.5 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 within those domains, such as consumer-facing software products.

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 three 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 were confounded by our low inter-rater reliability score on the PostgreSQL data-set, suggesting that annotators need careful and thoughtful training. With appropriate rigor in annotation, these results, along with the performance of our learners, demonstrate that labelled topic extraction can be a promising approach for understanding the occurrence of non-functional requirements in software projects.

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

References

1. Baldi, P.F., Lopes, C.V., Linstead, E.J., Bajracharya, S.K.: A theory of aspects as latent topics. In: Conference on Object Oriented Programming Systems Languages and Applications, pp. 543–562. Nashville (2008)
2. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet Allocation. Journal of Machine Learning Research **3**(4-5), 993–1022 (2003). DOI 10.1162/jmlr.2003.3.4-5.993
3. Bøegh, J.: A New Standard for Quality Requirements. IEEE Software **25**(2), 57–63 (2008). DOI 10.1109/MS.2008.30
4. Boehm, B., Brown, J.R., Lipow, M.: Quantitative Evaluation of Software Quality. In: International Conference on Software Engineering, pp. 592–605 (1976)
5. Chung, L., Nixon, B.A., Yu, E.S., Mylopoulos, J.: Non-Functional Requirements in Software Engineering, *International Series in Software Engineering*, vol. 5. Kluwer Academic Publishers, Boston (1999)
6. Cleland-Huang, J., Settimi, R., Zou, X., Solc, P.: The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. In: International Requirements Engineering Conference, pp. 39–48. Minneapolis, Minnesota (2006). DOI 10.1109/RE.2006.65
7. Ernst, N.A., Mylopoulos, J.: On the perception of software quality requirements during the project lifecycle. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Essen, Germany (2010)
8. Fawcett, T.: An introduction to ROC analysis. Pattern Recognition Letters **27**(8), 861 – 874 (2006)
9. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press (1998)
10. Few, S.: Information Dashboard Design: The Effective Visual Communication of Data, 1 edn. O'Reilly Media (2006). URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0596100167`
11. Flach, P.: The geometry of roc space: understanding machine learning metrics through roc isometrics. In: Proc. 20th International Conference on Machine Learning (ICML'03), pp. 194–201. AAAI Press (2003). URL `http://www.cs.bris.ac.uk/Publications/Papers/1000704.pdf`
12. Forman, G., Scholz, M.: Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. SIGKDD Explor. Newsl. **12**, 49–57 (2010). DOI http://doi.acm.org/10.1145/1882471.1882479. URL `http://doi.acm.org/10.1145/1882471.1882479`
13. German, D.M.: The GNOME project: a case study of open source, global software development. Software Process: Improvement and Practice **8**(4), 201–215 (2003). DOI 10.1002/spip.189
14. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations **11**(1), 10–18 (2009). URL `http://www.kdd.org/explorations/issues/11-1-2009-07/p2V11n1.pdf`
15. Hindle, A., Ernst, N.A., Godfrey, M.W., Mylopoulos, J.: Automated topic naming to support cross-project analysis of software maintenance activities. In: International Conference on Mining Software Repositories (2011)
16. Hindle, A., German, D.M., Holt, R.: What do large commits tell us?: a taxonomical study of large commits. In: MSR '08: Proceedings of the 2008 international working conference on Mining software repositories, pp. 99–108. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1370750.1370773
17. Hindle, A., Godfrey, M.W., Holt, R.C.: Release Pattern Discovery via Partitioning: Methodology and Case Study. In: International Workshop on Mining Software Repositories at ICSE, pp. 19–27. Minneapolis, MN (2007). DOI 10.1109/MSR.2007.28
18. Hindle, A., Godfrey, M.W., Holt, R.C.: What's hot and what's not: Windowed developer topic analysis. In: International Conference on Software Maintenance, pp. 339–348. Edmonton, Alberta, Canada (2009). DOI 10.1109/ICSM.2009.5306310
19. Software engineering – Product quality – Part 1: Quality model. Tech. rep., International Standards Organization - JTC 1/SC 7 (2001)
20. Kayed, A., Hirzalla, N., Samhan, A., Alfayoumi, M.: Towards an ontology for software product quality attributes. In: International Conference on Internet and Web Applications and Services, pp. 200–204 (2009). DOI 10.1109/ICIW.2009.36

21. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: International Joint Conference On Artificial Intelligence, pp. 1137–1143. Toronto (1995). URL http://portal.acm.org/citation.cfm?id=1643047
22. Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J.: An information retrieval approach to concept location in source code. In: 11th Working Conference on Reverse Engineering, pp. 214–223 (2004). DOI 10.1109/WCRE.2004.10
23. Massey, B.: 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)
24. McCall, J.: Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager, vol. 1-3. General Electric (1977). URL http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA049055
25. Mei, Q., Shen, X., Zhai, C.: Automatic labeling of multinomial topic models. In: International Conference on Knowledge Discovery and Data Mining, pp. 490–499. San Jose, California (2007). DOI 10.1145/1281192.1281246
26. Mockus, A., Votta, L.: Identifying reasons for software changes using historic databases. In: International Conference on Software Maintenance, pp. 120–130. San Jose, CA (2000). DOI 10.1109/ICSM.2000.883028. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=883028
27. Scacchi, W., Jensen, C., Noll, J., Elliott, M.: Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes. In: International Conference on Open Source Systems, vol. 1, pp. 1—-8. Genoa, Italy (2005)
28. Treude, C., Storey, M.A.: ConcernLines: A timeline view of co-occurring concerns. In: International Conference on Software Engineering, pp. 575–578. Vancouver (2009)
29. Tsoumakas, G., Katakis, I., Vlahavas, I.: Mining multi-label data. In: O. Maimon, L. Rokach (eds.) Data Mining and Knowledge Discovery Handbook, 2nd edn. Spring (2010)