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

Do Topics Make Sense to Managers and Developers?

Abram Hindle · Christian Bird · Thomas Zimmermann · Nachiappan Nagappan

the date of receipt and acceptance should be inserted later

Abstract Large organizations like Microsoft tend to rely on formal requirements documentation in order to specify and design the software products that they develop. These documents are meant to be tightly coupled with the actual implementation of the features they describe. In this paper we evaluate the value of high-level topic-based requirements traceability and issue report traceability in the version control system, using *Latent Dirichlet Allocation* (LDA). We evaluate LDA topics on practitioners and check if the topics and trends extracted match the perception that industrial Program Managers and Developers have about the effort put into addressing certain topics. We then replicate this study again on Open Source Developers using issue reports from issue trackers instead of requirements, confirming our previous industrial conclusions. We found that efforts extracted as commits from version control systems relevant to a topic often matched the perception of the managers and developers of what actually occurred at that time. Furthermore we found evidence that many of the identified topics made sense to practitioners and matched their perception of what occurred. But for some topics, we found that practitioners had difficulty interpreting and labelling them. In summary, we investigate the high-level traceability of requirements topics and issue/bug report topics to version control commits via topic analysis and validate with the actual stakeholders the relevance of these topics extracted from requirements and issues.

Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Canada
E-mail: abram.hindle@softwareprocess.es

Christian Bird and Thomas Zimmermann and Nachiappan Nagappan
Microsoft Research
Redmond, WA, USA
E-mail: {cbird,tzimmer,nachin}@microsoft.com

1 Introduction

For many organizations requirements and specifications provide the foundation for the products that they produce. As requirements are implemented the links between requirements and implementation weaken, especially during maintenance. Later, development artifacts often stop referencing the requirements documents that they were derived from. Current research shows that there is a lack of traceability between requirements and implementation [20] whereas the managers we interviewed expected and wanted requirements and implementation to be in sync (Section 5.2). The volume of traceability research confirms its importance [8, 32, 38, 39]. In this paper we extract topics from a large body of requirements documents and then search for commits that mention these topics within the version control system. These topics are represented as word distributions. These topics provide some high-level traceability between requirements and implementation once they are labelled and interpreted. Yet these topics can be exploited to provide an overview of development effort relevant to each topic.

In this paper we attempt to validate these topic-generated overviews by asking industrial developers, industrial program managers, and *Free/Libre Open Source Software* (FLOSS) developers if their perception of their own behaviour matches the behaviour highlighted by the topic. We do this by relating the topics extracted from requirements and issue reports to the commit log messages in version control systems. Thus we seek to validate if topics extracted from requirements and issue reports make sense to practitioners.

Stakeholder based validation of topics in terms of relevance, labelling, and the recovery of behaviour [3] is critical, but to date has not been widely applied to the domain of software engineering [14]. We also ask how well non-experts, such as the authors of this paper, can label topics on projects that we did not write. The topics we study are extracted using Latent Dirichlet Allocation [5] (LDA) which has gained popularity in software engineering (SE) research [3, 9, 12, 13, 15, 22, 27, 36]. Our contributions include:

- A technique for linking requirements to code commits via topics.
- An evaluation of the relevance of topics extracted by LDA from requirements with developers and managers.
- An analysis of whether topic highlighted behaviour matches the perception of industrial developers and managers as well as FLOSS developers.
- Insight into the difficulties that practitioners face when labelling topics and the need for labelled topics.
- Validation of non-expert topic labelling with practicing experts.
- A FLOSS developer oriented replication of the industrial study on issue tracker topics.

We are investigating if LDA topics make sense to practitioners, and whether practitioners can label LDA topics. We have at our disposal many skilled Microsoft developers who work on a very large software system that has many requirements documents. We also have the participation of 13 gracious and

skilled *Free/Libre Open Source Software* (FLOSS) developers associated with 13 different FLOSS projects, each with issue trackers and version control systems. Thus we investigate if practitioners face difficulties interpreting topics, if unlabelled topics are enough, and if familiarity with the source and domain of the topics matters.

1.1 Motivation

Latent Dirichlet Allocation (LDA) [5] and Latent Semantic Indexing (LSI) [23] are popular tools in software engineering research [4,33,36,37]. They are often used for traceability and information retrieval, but their use is built upon assumptions regarding usability. We validate this usability of topics assumption with managers and developers from our prior work [14] and FLOSS developers in this extension. Often in SE literature these topics are interpreted solely by the researchers themselves (e.g., [15]). Thus the use of topic analysis through algorithms like LDA and LSI in software engineering has not been widely and rigorously validated with actual developers and managers. The core motivation of this work is to validate: if topic analysis (with LDA) of requirements documents produces topics relevant to practitioners; if the extracted topics make sense to software developers; and if the development behaviour associated with a topic matches the perception shared by the practitioners. This work could be used in traceability tools, project dashboards for managers, effort models, and knowledge management systems.

In prior work we felt that topics needed to be labelled to be useful [15]. We were motivated by the belief that labelled topics allow stakeholders such as managers to track development efforts related to these extracted topics. We sought to evaluate the effectiveness of depicting document relevance to topics over time as *topic-plots* (see Figure 2 for an example). Thus we ask stakeholders to help verify if combining topics with commits allows for local analysis of requirements-relevant effort of different groups of developers or teams. An example scenario we are targeting would be a manager who is trying to identify how much effort went into addressing different requirements. Managers could try to answer the question, "Who should I talk to regarding a requirement change in the bluetooth stack?" by using individual developer topic-plots and correlating the developer's visible effort within a bluetooth related topic.

1.2 Extension and Replication

This work differs from our previous ICSM 2012 publication [14] because we have extended it with a FLOSS-oriented replication on issue-tracker topics rather than requirements topics.

We repeated most of the industrial methodology with the FLOSS developers: we sought out FLOSS developers and presented them with a very similar

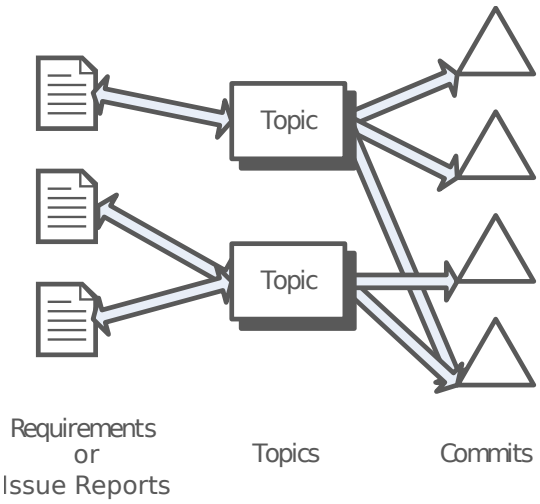


Fig. 1: Our data model. Topics are extracted from Requirements using LDA. Topics are related to requirements in a many-to-many relationship. Commits are related to topics via LDA document-topic inference.

survey. We then aggregated the results and compared. Thus the field work associated with the FLOSS survey was equivalent or larger in scale than the industrial case study.

In this paper, we added a comparison between the case studies and discussed the results of the FLOSS study in depth. For the sake of brevity we have combined the methodologies into one general description and included special cases for the FLOSS study when necessary.

2 Background and Previous Work

Our work fits into traceability, requirements engineering, issue tracker querying, and topic analysis [7, 8, 22, 32, 36].

2.1 Traceability and Requirements Engineering

Traceability is the linking of software artifacts and popular software engineering topic. Authors such as Ramesh et al. [30] and De Lucia [10] have provided excellent surveys of the literature and the techniques relevant to traceability. In terms of *information retrieval* (IR) and traceability Antonional et al. [1] first investigated linking documentation and source code together using IR techniques including the vector space model. IR traceability was extended by Marcus et al. [23] who first employed LSI for traceability of documentation to source code. Karl Wiegers [39] has argued for tagging commits with relevant requirements to aide traceability. Tillmann et al. [38] have discussed mining

specifications by their language in order to aid traceability for reverse engineering. Kozlenkov and Zisman et al. [20] have also studied requirements traceability with respect to design documents and models. Ernst et al. [11] traced nonfunctional requirements (NFRs) within version control systems (VCSs) using a simple technique incorporating NFR word dictionaries. Murphy et al. [26] defined explicit mappings between concepts and changes to files but did not use topics; whereas Sneed [35] investigated mining requirements in order to produce test cases. Reiss et al. [31] produced a tool called CLIME that allows one to define constraints and track the co-evolution of artifacts in order to enforce adherence. Poshyvanyk et al. [25, 28, 33] has explored the use of IR techniques for software traceability in source code to other kinds of documents. Others within the RE community have leveraged natural language processing (NLP) techniques to produce UML models [19]. NLP techniques and IR techniques, such as n-grams, stemming, bag of words models, and vector space models are used in conjunction with topic analysis.

2.2 Topics in Software Engineering

Topics in software engineering literature are known by many names: concerns, concepts, aspects, features, and sometimes even requirements. In this paper, by *topic* we mean a word distribution extracted from requirements documents by an algorithm such as Latent Dirichlet Allocation (LDA) [5] that often matches a topic of discussion between authors. LDA helps us find topics by looking for independent word distributions within numerous documents. These documents are represented as word distributions (i.e., counts of words) to LDA. Given a number n LDA then attempts to discover a set of n topics, n word distributions that can describe the set of input documents. Each document is then described as a combination of the n topics that are extracted. Thus the end result of topic analysis is a set of n topics (word distributions) in the form of a the *word-topic matrix*, and a *topic-document matrix* that provides the relationship between documents and topics.

Documents are not mutually exclusive to topics. This means that 1 document can be related to 0 or 1 to n topics. The allocation part of Latent Dirichlet Allocation implies that words and documents are allocated to topics. But since they can be partially relevant to a topic they are also partially allocated. But the allocation implies there is a limit, thus the topics and words are shared among topics. For example, if a document is allocated solely and equally to 2 topics the document's row in the topic-document matrix will have half of its "space" allocated to one topics and the other half allocated to the second topic. If this "space" was represented as probability, each entry would be 0.5; if this "space" was word counts then the word counts would be equal. Thus while a document can be allocated to many topics, there is a limit to how much of a document or word can be allocated to a particular topic. Fundamentally it means that documents are often allocated to more than one topic.

Since each LDA topic is a word distribution over many words, we must present an alternative representation to end-users such as developers and managers. These topics can be represented to end-users as a ranked list of words, from highest magnitude to lowest magnitude word relevance. Many researchers use top-10 lists of words, in this study we used 20 words. An example topic might be:

code improve change without functionality behaviour readability maintainability structure restructure modify reduce quality process complexity software re-factoring performance maintain better

How would you label this topic? Notice how this topic takes time to interpret. In this paper we investigate the difficulty practitioners have when labelling the topic as well as the relevance of the topic to practitioners.

There is much software engineering research relevant to topics. Many techniques are used, ranging from LDA [15] to Latent Semantic Indexing (LSI) [24]. Researchers such as Poshyvanyk et al. [28] and Marcus et al. [24] often focus on the document relationships rather than the topic words. In terms of topics and traceability, Baldi et al. [4] labelled topics and then tried to relate topics to aspects in software. Asuncion et al. [3] used LDA on documentation and source code in order to provide traceability links. Gethers et al. [12] combine IR techniques such as Jenson and Shannon, vector space model, and relational topic model using LDA, together into one integrated in order to aide traceability link recovery. They found this integration of techniques achieved better performance than any technique alone.

Grant et al. [13] have worked with LDA and topics before. Their 2010 paper suggests heuristics for determining the optimal number of topics to extract. Thomas et al. [36] statistically validated email to source code traceability using LDA. Panichella et al. [27] described LDA-GA, a genetic algorithm approach to search for appropriate LDA hyper-parameters and parameters. They evaluate these parameter choices against a range of software engineering tasks and thus evaluate the cost-effectiveness of this search approach. Both groups did not validate their results with practitioners.

Our work builds up on the work of Lukins et al. [22] as they apply topic analysis to issue reports as well, but use the topics to query for existing issue reports.

In this study we investigate human-generated labels for topics; while other researchers have investigated automatically generated labels for source code artifacts. De Lucia et al. [9] investigated using IR methods on source code artifacts in order to label software artifacts. They used multiple IR approaches, including LDA and LSI to see if they could label or summarize source code and if their approach matched human-generated labels. They found that labelling via simple heuristics best matched practitioner labels rather than LSI or LDA.

These studies rely on assumptions about the applicability to practitioners. We investigate some of these assumptions by surveying practitioners in order to validate the value of topics extracted from requirements. Furthermore instead of tracing individual requirements documents or sections of said documents

directly, we extract topics from sets of requirements and then track those topics within the version control's change history. Original requirements can be related to code changes by the topic-document association matrix as described in Figure 1. Next we describe our methodology.

3 Methodology

Our goal is to relate commits to topics of requirements and then validate if these topics and plots of topic-relevant commits make sense to stakeholders such as developers and managers. Our methodology is to extract requirements and issues, perform topic analysis on the documents and then infer and link these topics across all of the commit log messages in the source code repository. Then we present extracted topics to developers and managers and ask them to label the topics. After that, we show plots of topic-relevant commits to developers and managers and ask whether or not these plots actually match their perception of their effort related to the topic. Finally we analyze and present the results.

3.1 Requirements Mining

Microsoft stores requirements documents in their documentation repository. Requirements for a project are grouped by broad general topics. The documents are usually saved as Word documents within the specification repository. These requirements are usually written by program managers (PMs), developers and testers.

We obtained all of the network component specifications from a popular Microsoft product that is used by millions of users and has several millions of lines of source code. We then saved each of these specifications as text files for later analysis. This large collection of requirements documents included 75 total requirements documents consisting of nearly 1 500 pages of documentation, 59 000 lines of text, 35 000 paragraphs, and 285 000 words. The average document was 20 pages long with an average word count of 3 800.

Each requirements document has a title, a list of authors and small table of major revisions. The documents are then further broken down into sections much like IEEE documentation such as Software Requirements Specification (SRS) (the functional specifications), Software Design Description (SDD) (referred to internally as "dev specs"), and Software Test Documentation (referred to internally as "test specs"). Program managers that we interviewed indicated that requirements were generally written by the managers and the "dev specs" and "test specs" were written by developers and testers. The requirements were often the result of "quick specs" that had become "final specs" via a process of comment elicitation and discussion. Once a requirements document became a "final spec", it could be scheduled for a milestone and assigned to a team.

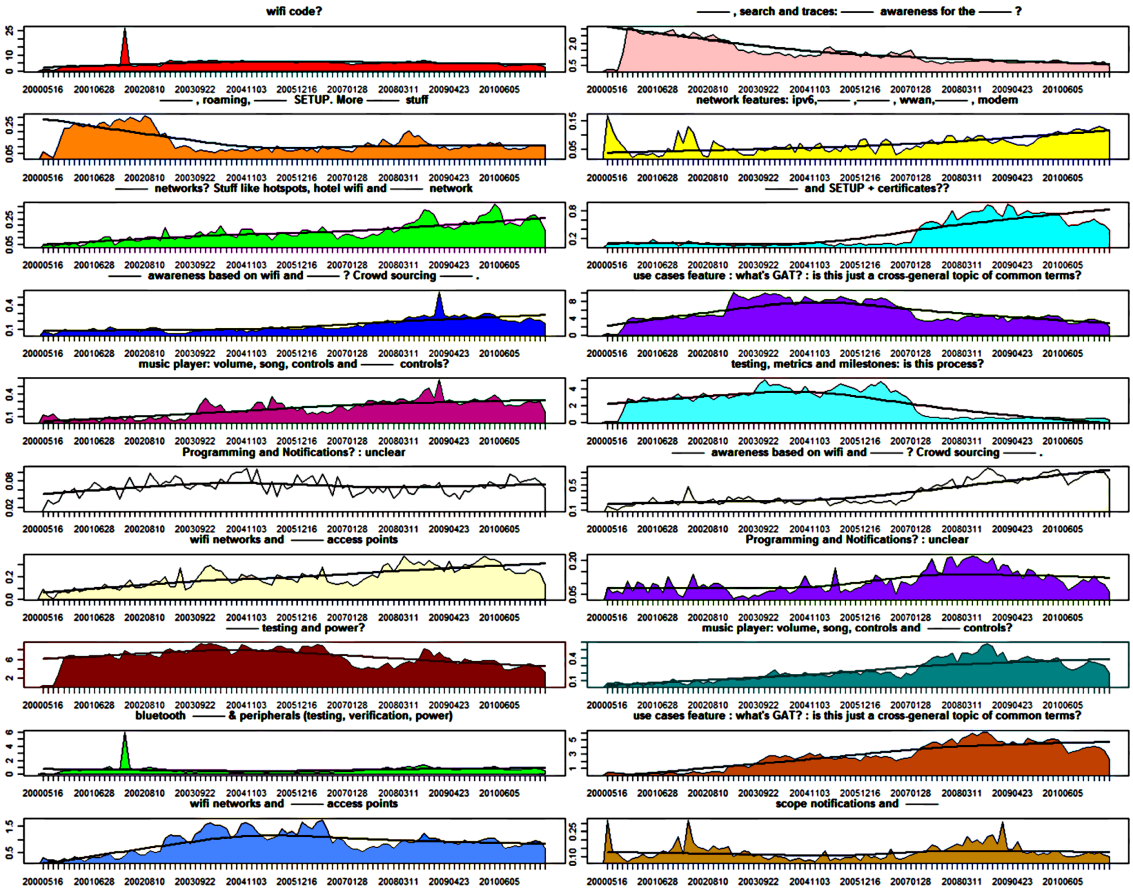


Fig. 2: Requirements Topic-Plots: Topics and Relevant revisions over time. X axis is time, Y axis is the average relevance per time window (greater is more relevant). The topic-plots are labelled with our non-expert labels. They are non-expert labels because we were not involved in the project's development. —— indicate redactions.

3.2 Requirements Topic Mining

To apply LDA to requirements we had to preprocess the documents. We converted each specification to word distributions (counts of words per document) and removed stop words (common English stop words such as "at", "it", "to", and "the") [1]. We stemmed the English words using a custom Porter stemmer. We provided these word distributions to our implementation of the LDA al-

[1] The set of stop words used: `http://softwareprocess.es/b/stop_words`

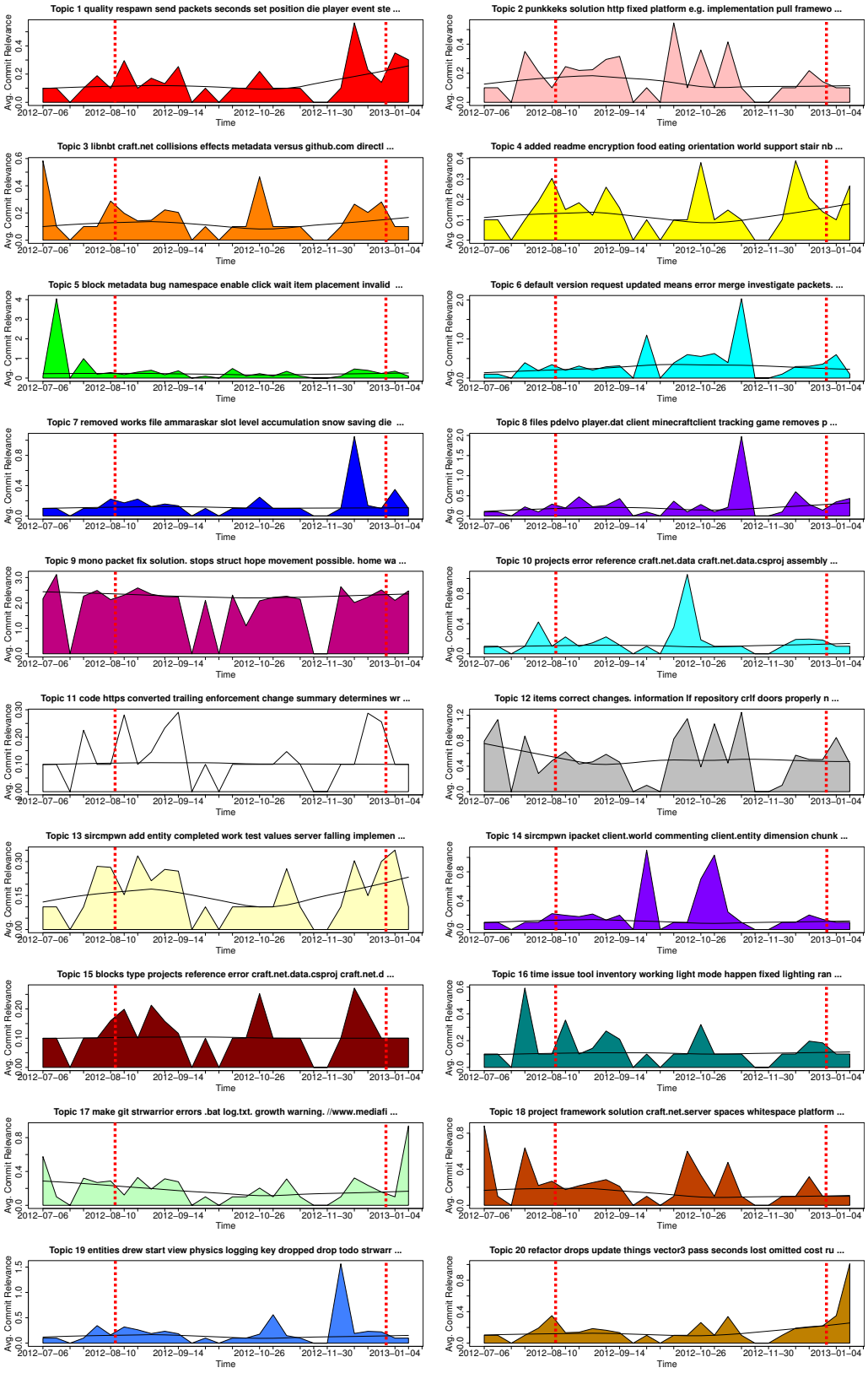


Fig. 3: Issue Topic-Plots of All FLOSS Developer authors of Craft.NET. X axis is time, Y axis the average per time window (greater is more relevant). Red dashed lines indicate a release or milestone of importance (major or minor). Topic Labels are the top ranked topic words from LDA.

gorithm. Then LDA processed the requirements documents and produced the requested number of topics.

The implementation of LDA we used was built by Microsoft and it was a parallel implementation of CVB0, a collapsed variational Bayes LDA implementation [2, 29]. LDA was configured hyper-parameters of $\alpha = 0.1$ and $\beta = 0.1$, and 1 000 iterations were used to extract topics from the requirements documents.

While authors such as Grant et al. [13] have proposed heuristic methods of suggesting the best number of topics, their idea relies on fitness functions that are not necessarily human-centric. Thus we took a more subjective approach. This subjective, manual, human-approach was taken because we were not convinced that silhouetting or genetic algorithms [27] would be appropriate given our lack of a topic coherency or topic readability oracle. We instead decided to act as this oracle, aiming for distinct and readable topics. This means we were biased to a fewer number of topics based on the effort we spent to interpret each topic. Thus we made a decision to ignore automated methods of determining the optimal number of LDA topics and instead opted for a subjective search based on qualities of semantic coherency and topic readability/interpretability.

To determine the number of topics to use, we ran LDA multiple times, generating 5, 10, 20, 40, 80, and 250 topics. We then chose the number where the extracted topics were distinct enough. This meant the first author applied his own judgment to ensure that topics did not have much overlap in top terms, were not copies of each other in terms of top words, and did not share excessive disjoint concepts or ideas. Conceptual overlap required human readers who understood some of the semantics of the words. Our number of topics selection process was like a manually applied fitness function with the first author evaluating the fitness of each set of topics.

Based on these requirements 20 topics seemed to produce the most optimal topics given our previous requirements. Thomas et al. [37] reported similar results. We argue that if practitioners were following this methodology they would not want many topics because it takes time to label each topic, as we personally noticed from this tuning step. This was later confirmed by our survey respondents took approximately 1 to 4 minutes (2 on average) for each topic, as discussed ahead in Section 7. Thus we used LDA to produce 20 topics.

3.2.1 Labelling of Requirements Topics

Once the topics were extracted from the requirements documents we then labelled each of the topics to the best of our knowledge by reading the top ranked topic words (we kept all topics words) and tried to label them using our non-expert domain knowledge. Only one author, the first author, labelled the topics. We refer to these topics labels as *non-expert* labels as the first author did partake in the development of the project being studied. Labelling topics was difficult as there were many project specific terms that did not have a clear definition (such as GAT, setup, and RIL).

The motivation behind labelling topics is that they are time consuming to interpret and are faster to reason about if labelled. Furthermore we wished to compare our labels to those of domain experts (the relevant developers).

3.3 Version Control Mining

To correlate topics and development activity we extracted the change log messages from 650 000 version control system commits of a popular Microsoft product. We had approximately 10 years' worth of commits from more than 4 000 unique authors. Our properties per commit consisted of user name, machine name, branch name and the change description (also known as the commit log message).

For the FLOSS projects we had to mine a variety projects listed in Table 1. Note that Table 1 names the FLOSS participants directly because those participants chose to self-identify; they were given the choice of anonymity, project-level anonymity, or full identity with attribution. All of the FLOSS participants chose full identity with attribution. Those on Google Code tended to use SVN so we used `git-svn` [2] to convert SVN to Git. Git repositories were processed using a custom script, `git-grep.pl` [3] that extracted the commits into a common JSON [4] format.

3.4 Issue Tracker Topic Mining

We mined issue trackers of both Github [5] and Google Code [6] using our own issue tracker mining programs.

We extracted their issues and the comments associated with the issues and converted them into a custom, but common schema in a JSON format.

To extract LDA topics from issues extracted from Google Code or Github we followed much of the same methodology in Section 3.2.

Much like in Section 3.2 we removed stop words from the texts (for the stop words used please see Footnote 1 in Section 3.2). We were not able use the same stemmer for this system so we did not stem the issues at all. There was no identifier splitting applied either. Also the texts were composed of the author, owner, subject and bodies of an entire issue, joined into a single document concatenated with the author and comment bodies of the associated issue comments (the discussion of the issue in the issue tracker). This is because the requirements contained similar information about authorship in-lined in

[2] git-svn man page: `https://www.kernel.org/pub/software/scm/git/docs/git-svn.html`

[3] `git-grep.pl` is located here: `https://github.com/abramhindle/gh-lda-extractor`

[4] JSON Definition: `http://JSON.org`

[5] Github Issue Extractor: `https://github.com/abramhindle/github-issues-to-json`

[6] Google Code Issue Extractor: `https://github.com/abramhindle/google-code-bug-tracker-downloader`

their texts. Since users and developers could get into the industrial topics we thought it was necessary to emulate requirements documents by including authorship information. These documents were tokenized and fed into LDA. We did not filter out any issue tracker documents.

We used a different implementation of LDA for the FLOSS issue tracker study, Vowpal Wabbit. [7] The source code for the FLOSS issue tracker part of the study can be found on-line[8] (Vowpal Wabbit is required).

209 projects were extracted, and 13 eventually used; many of which were small. To maintain consistency with the industrial case study the same number of topics were extracted, $n = 20$. The minimum requirement of a project was that it needed at least 20 issues in order to provide enough data to make topics from. Vowpal Wabbit (VW) was configured with LDA hyper-parameters $\alpha = 0.1$, $\beta = 0.1$, and 2 online iterations. VW uses an online algorithm for LDA, thus iterations are done in batches. Vowpal Wabbit uses a variational Bayes LDA algorithm [16]. Figure 3 shows the results of extracting and plotting topics of Craft.NET.

3.5 Relating Requirements Topics to Commits

To relate commits to requirements topics we used LDA inference. LDA inference is similar to how LDA trains and learns topics except it does not learn from this inference — it relates documents to pre-existing topics. This allows us to reuse the existing requirements topics. LDA inference takes topics and documents as input and produces a new topic-document matrix (see Section 2.2) that represents the association of a document (a commit message in our context) to each of the 20 topics we had already extracted from requirements. LDA inference allows us to relate existing topics to new documents without modifying the existing topics. A new topic-document matrix is created that describes this topic-document inference. The LDA inference technique was previously used by Lukins et al. [22] to query bug reports.

In order to relate commits to requirements topics that were already generated, we had to convert the requirements topics to word distributions and then infer the relationship between their word distribution and the topic word distribution via LDA inference, rather than rerunning LDA to produce new topics. Thus we tokenized the commit log message of each commit and produced a word distribution per commit. We treated these documents in the same manner as the requirements documents: we removed stop words, stemmed the terms with a custom Porter stemmer, and used the intersection of the vocabulary shared by the requirements documents and topics. We intersected the commit and requirements vocabulary because we were using LDA inference and thus were not learning new topic words. We did not split words, or split identifiers, as we were worried about adding semantics or removing seman-

[7] Vowpal Wabbit: `https://github.com/JohnLangford/vowpal_wabbit/wiki`

[8] Our Github LDA Extractor: `https://github.com/abramhindle/gh-lda-extractor`

tics since these topics might be interpreted by experts. The commits were not filtered by the quality or length of their commit messages.

Thus we intersected each commit's words by the words in our topics and then inferred (via *LDA inference*) the relationship between the requirements topics and the changes over time. We inferred the topics related to each change, leaving us with a topic-document matrix of changes associated with topics. Figure 1 depicts this relationship between the version control system, LDA topics and the requirements documents. Topics are extracted from and related to requirements documents and then the relationship between topics and commits is inferred. We did not train on the commits because our goal is to use topics that were requirements relevant. Also, inference allows for fewer topic updates as requirements updates are less frequent than commits. This can allow us to plot time-series of commits that are relevant to topics. In Section 3.7, after the next section, we discuss how this matrix allows us to plot the relationship over time between requirements topics and commits. In the next section we discuss how apply this methodology to issue tracker topics.

3.6 Relating Issue Tracker Topics to Commits

To relate issue tracker topics to commit we follow the same methodology as Section 3.5 except our LDA corpus extracted from an issue tracker instead. In error we failed to apply stemming to relate issue tracker commits to issue tracker issues, and this was not correctable as the surveys had already been sent. The inference is exactly the same except we use Vowpal Wabbit for the LDA inference implementation and we did not stem words. To remain consistent with the industrial case study 20 topics were used, except 20 topics were extracted from each of the 13 FLOSS project's issue tracker. Extraction of issue reports is described in Section 3.4. Commits were not filtered based on the length of the commit log message or the quality of the commit log message.

3.7 Topic-Plots of Effort Relevant to Topics of Requirements

In order to communicate the effort that was associated with a requirements topic and the related requirements documents, ones has to present summaries of this effort to the users. We used a time-line overview, a topic-plot (see Figure 2) that shows the relevant commits over time. We also utilize our non-expert topic labels to label these plots as shown in Figures 2 and 4. These topic-plots can be generated at various levels of granularity: entire projects, entire teams, or individual developers. Using one single global analysis we can select subsets of the commits by attributes such as time, topic, author, or team in order to produce more localized topic relevance plots. In particular we found that developers were more likely to remember their own effort thus we produced topic-plots based solely on their own commits; we called these plots *personal*

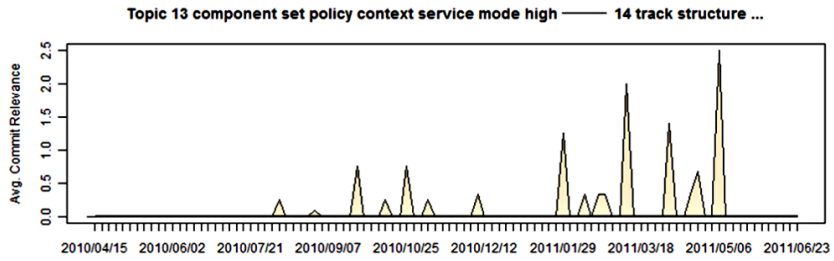


Fig. 4: Personal topic from 1 industrial developer. Topics and Relevant revisions over time. X axis is time, Y axis is the average relevance per time window (greater is more relevant). The topic label provided is a non-expert label created by the authors.

topic-plots. Figure 4 is an example of one of these personal topic-plots. These topic-plots are appropriate representations as they provide stakeholders with an overview of topic relevant effort and would be a welcome addition to a project dashboard.

In this study we assume that commits are a reasonable proxy for effort. We recognize that commits could actually be months of work, expressed as 1 sole commit, or many commits could be created in the same hour. Thus we look to the prior work of Koch et al. [18] and Capiluppi et al. [6] who argue and provide evidence that commits are correlated with project activity and are both related and correlated with development effort. We argue that the topic relevance of a commit is relevant to the effort related to that topic and the associated documents that the topic was extracted from. While it might not be an exact measure, a strong relevance would indicate that potentially similar concepts are being expressed in the commit comments that are relevant to the topic at hand. Thus we argue that not only are commits relevant to effort, but that topic relevant commits indicate topic relevant effort.

Note that a single commit can be associated with multiple topics [13]. Figure 2 shows a topic-plot of 20 topics that show the focus on these topics by average commit relevance over time. We put the commits into bins of equal lengths of time and then plot their average relevance (a larger value is more relevant) to that topic. Bins will be of an equal length of time (such as 7 days) and thus they will not be equal size in terms of the number of commits. Some bins will have few or 0 commits, some will have 10s to 100s of commits depending on the project and how busy development was at that time. Then for the commits within a bin, their topic-document relevance values from the topic-document matrix, extracted by LDA inference, will be averaged together. The average relevancy of 0 commits is set to 0.

Note that —— indicates a redaction, within this study we redacted some tokens and words that would disclose proprietary information. This is the

long dash that is featured in Figure 2. Within this figure we can see distinct behaviours such as certain topics increasing or decreasing in importance.

The figures produced are interesting but we have to evaluate if they are representative of the developer's perception of their relevant behaviour.

3.8 Topic-Plots of Effort Relevant to Issue Tracker Topics

For the Issue Tracker Topic-Plots we followed a similar methodology to the previous Section 3.7. But we executed this methodology per each project and per each author of that project.

Figure 3 shows the topic-plots of Craft.NET, while Figures 6 and 7 show the topic-plots given to 13 different FLOSS developers on 13 different FLOSS projects.

3.9 Qualitative Evaluation

Our goal is to determine the utility of the global revision topic-plots and the personal topic-plots (e.g. Figure 4). Thus we interviewed relevant stakeholders and developed a survey. Our interviews and surveys were designed using methodologies primarily from Ko et al. [17] and those described by the empirical software engineering research of Shull et al. [34] and Wohlin et al. [40], regarding interviews, surveys and limited study sizes.

3.9.1 Microsoft PM and Developer Interviews

Our goal is to determine if labelled topics are relevant to developers and program managers (PMs), all of whom are development engineers. Our first study occurred at Microsoft and thus the initial interviews were with Microsoft employees. Relevance to a developer is subjective, but we define it as "the developer worked on a task that had at least a portion of it described by the topic". We scheduled and interviewed 1 PM and 1 developer for 1 hour each. During the interviews with PMs and developers we asked if the topic-plots of the 20 requirements topics were relevant to developers and PMs. We did not provide our labels to the PMs and developers until after they had provided labellings. We also asked, "Are they surprised by the results?" and, "Can they identify any of the behaviours in the graphs?" We then summarized the comments and observations from these interviews and used them to design the survey, discussed in the next section.

3.9.2 Microsoft Developer Survey

To gain reliable evidence about the utility of requirements topics and topic-plots, we produced a survey to be administered to developers, that asked

developers to label topics and to indicate if a personalized plot of their behaviour matched their perception of their own effort that was related to that topic and its keywords. We analyze these results in the Section 5.2.

The survey asked developers to label three randomly chosen topics, evaluate the three personal topic-plots of these topics, and to comment about the utility of the approach. Each of the topic labelling questions presented an unlabelled topic with its top 20 words and asked the respondents to label the topic. Another question asked if this topic was relevant to the product they worked on. They were then asked to look at a personal topic-plot and see if the plot matched their perception of the effort they put in. Finally they were asked if these plots or techniques would be useful to them. The surveys were built by selecting the developer's commits and randomly selecting topics that the developer had submitted code to. Three topics were randomly chosen for labelling and these topics were ordered randomly in an attempt to limit ordering effects.

Initially, we randomly selected 35 team members who had committed changes within the last year and had over 100 revisions. We sent these developers an email survey. After only 4 developers responded we switched to manually administering surveys. We repeated the previous methodology for choosing participants and chose 15 candidates, 8 of whom agreed to be administered the survey. Administering the surveys in person would reduce training issues and allow us to observe more. The administered surveys included an example topic question, used for training, where we verbally explained how the topics were extracted and how the commit messages were analyzed during the in-person administration of the survey. We then walked them through this example, reading the topic words out loud and thinking out loud. We provided a concrete example and abstract explanation to increase survey success or completion.

Surveys were administered by the first two authors, the first author spoke aloud each question to the respondents in order for their answers to conform to the survey. An example of a labelling that perceptually matched was when a respondent gave the label "Design/architecture words" to the following Topic 8 (seen in the 2nd column of the 4th row of Figure 2):

component set policy context service mode high —— 14 track structure check type hresult feature gat ap follow data dynamic.

Later, similar surveys were produced for FLOSS developers.

3.9.3 FLOSS Developer Survey

We sought to replicate the Microsoft case study, described in Section 3.9.2, that was executed on industrial participants, on FLOSS Developers.

The FLOSS survey was meant to be shorter than the MS survey since the FLOSS survey had a lengthy consent form attached to it. Much like the original survey the FLOSS developer survey asked developers to label three randomly chosen topics, determine if these topics were relevant to their project, and

evaluate if three personal issue tracker topic-plots presented matched their perception, and then they were asked to comment about why or not the plot did not match their perception. We did not replicate one set of questions: we did not ask FLOSS developers about the utility of the plots in the survey because we assumed that we would have a higher rate of interview participation and thus we could go beyond just yes/no answers. Furthermore we needed to keep the time to fill out the survey low due to the lengthy consent form. We found in the Microsoft study that a simple yes/no question was probably oversimplifying the response of the survey participants.

These surveys were built by selecting the FLOSS developer's commits and randomly selecting topics that the developer had submitted code to. Three topics were randomly chosen for labelling from 20 generated topics for their project. These topics were ordered randomly in an attempt to limit ordering effects if there were any.

The surveys were uploaded to a website, and later we added the surveys to Google Drive. We uploaded an HTML survey, an open document survey (LibreOffice/OpenOffice), a DOC file survey (Microsoft Word), a PDF survey, and sometimes a Google Doc survey. Each of these surveys were identical. Our hope was to be as convenient as possible for the respondents, some respondents mentioned that Google Docs was the most convenient. For the other file types the respondents would email back the modified files. We did not send HTML email to the FLOSS developers because there is some backlash in the FLOSS community against HTML email.

To find FLOSS developers we started by browsing Google Code and finding projects that were used but not exceptionally busy and popular. We then extracted their issue tracker and commits. Surveys and suggested emails were generated by an email/survey generator script and we emailed many of these off. After receiving limited response from the developers of Google Code projects we switched to Github projects. We then decided to approach developers more directly using real-time chat.

Our recruitment strategy was to join Freenode [9], a popular Opensource Internet Relay Chat (IRC) network, and look for channels that advertised Github repositories. Channels can advertise a repository by having the URL to the repository in their channel topic (e.g, the project github3.py had a channel topic of "https://github.com/sigmavirus24/github3.py current version: 0.5 || http://developer.github.com/v3/ || http://github3py.rtfd.org/"). We browsed the Freenode channels using the `/LIST` command in IRSSI [10]. Then we filtered the channels by those that had Github URLs in them. If their Github project had over 20 issues in the Github Issue Tracker and more than 3 months of development in the Github git repository we would enter the channel, announce that we have extracted the topics of their project and paste a hyperlink to a gallery of the extracted global issue tracker topic-plots. We selected the channels by number of participants and in alphabetical order. We started with

[9] FreeNode: `http://freenode.org`
[10] IRSSI IRC Client: `http://irssi.org/`

the lowest number of participants first as we expected the idle participant would probably be a developer for that project. Thus for each channel we joined, we had already mirrored their issue tracker and git repository, and extracted the issue tracker comments and pre-generated all the surveys for all the developers in the project's git repository.

We would target the known developers in the IRC channel by prefixing our gallery URL with their nicknames (e.g, "devname: hello! I've plotted the issue topic of your repository at: URL"). Then if a developer responded we would engage them in conversation and ask if they would interested in taking part in our survey. Of these Freenode developers, about 11 positively responded.

In total we had sent out 156 emails, this included the emails to the Freenode based developers. We had already extracted and generated surveys for 209 different projects. We received 13 responses. Our response rate overall was 12%, much lower than our Microsoft response rate but we had less developers per project to choose from. We found that the direct developer engagement resulted in more respondents. Some developers mention they had been contacted before by other researchers. Developers who responded, and their corresponding FLOSS projects, which had issues and commits, are listed in Table 1. Once the surveys were collected we summarized the results.

3.10 Summarizing the results

After the surveys were administered we discussed the results with each other, analyzed the data and collated our observations about the topic labelling, topic interpretation and if the topic-plots matched the developers' perception of effort. Figures 6 and 7 depict the issue report topic-plots from the FLOSS developers, while Figure 2 shows the requirements topic-plots of the Microsoft product that we studied. In the next sections we discuss the results of these interviews and surveys.

4 Topics of Requirements, Issues and Topic-Plots

We argue that leveraging these topics provides some traceability between requirements documents and the commits themselves. When one combines these links into a topic-plot (Figure 2) one gains a powerful high-level overview of the effort that is relevant to certain requirements.

Figure 2 depicts a set of topic-plots based on the 20 requirements topics that we extracted, and manually labelled, from the large Microsoft project that we studied. What is important about this style of plot is that it provides an overview of the per topic focus of effort during an entire system's lifetime. One can observe the evolution of how efforts are focused on different kinds of requirements topics at different times in the system.

The data being processing is large and cumbersome, yet such a simple plot can provide insights into a project. For instance, the 6th topic (3 down, right

FLOSS Developer	Project	Issues	Commits
Julian Harty	Open reader for DAISY 2.02 Audio Books https://code.google.com/p/android-daisy-epub-reader/ An Android EBook Reader	59	529
Lisa Milne	nodau https://github.com/darkrose/nodau A commandline based note taker	14	52
Tobias Leich	SDL for Perl https://github.com/PerlGameDev/SDL Perl bindings for the SDL Library	242	2 714
Ian Cordasco	github3.py https://github.com/sigmavirus24/github3.py Python bindings to Github's API	72	816
Ricky Elrod	dagd https://github.com/CodeBlock/dagd RESTful PHP Network Tools	13	220
Anthony Grimes	refheap https://github.com/Raynes/refheap A Clojure Pastebin-like Webservice	107	475
Geoffrey Greer	sublimetext2plugin https://github.com/Floobits/sublime-text-2-plugin Floobits plugin for Sublime Text 2	28	226
Nicolas J. Bouliane	DNDS https://github.com/nicboul/DNDS A Peer-2-Peer (P2P) Virtual Private Network (VPN)	28	1 065
Drew DeVault	Craft.NET https://github.com/SirCmpwn/Craft.Net MineCraft utility library for .NET	147	479
Daniel Huckstep	kindlebility https://github.com/darkhelmet/kindlebility/ Makes Readable (Readability) PDFs of Webpages for the Kindle	49	113
Chad Whitacre	Aspen https://github.com/zetaweb/aspen Pythonic Web Framework	160	1 243
Devin Joel Austin	Form-Sensible-Reflector-DBIC https://github.com/dhoss/Form-Sensible-Reflector-DBIC DB Schema based Form Generator	8	102
Gerson Goulart	Aura https://github.com/aurajs/aura Javascript UI Framework	216	367

Table 1 FLOSS Developers and their Projects

most), in Figure 2 shows a lack of behaviour for the first 7 years followed by a distinct and constant increase in focus and importance. In Section 5.1.1 an interview with a program manager reveals that the spikes in the topic-plots 7 and 9 are relevant to actual changes (topic-plots 7 and 9, 4th and 5th rows in the first column of Figure 2). Another example of interesting behaviour includes the use cases and testing topics (8th and 10th topics, 4th and 5th in the second column) became less important, information that would be useful to a manager.

Figure 2 is a global plot; while relevant to managers it might not be relevant to individual developers. Developers could be more interested in fine-

grained and local information about themselves, their teammates and their entire team. Example fine-grained and local topic-plots are depicted in Figure 5a which shows the topic-plots of 2 developers drawn only from commits made by those developers. By examining Figure 5a, we observe that the developers' behaviour with respect to a particular topic does indeed change over time, and each developer exhibits a different focus. Plots like Figure 5a illustrate that different authors' focus evolves over time.

Analysis of groups of developer via topic-plots is possible. Figure 5b depicts different teams (developers with the same manager) rather than different authors. These plots can be generated based on the work of multiple authors or an organization, in this case we leveraged the organizational knowledge of teams (who manages who). This allows the attribution of effort relevant to requirements topics to teams. For Topic *testing, metrics and milestones* in Figure 5b, the trend was similar but the behaviour was not. This manager level view provides a summary of a team's development effort and allows one to compare teams and see the behaviour relevant to that team.

In summary, we can track commits related to topics extracted from requirements over time. We are able to extract and plot global topic-plots depicting global trends, and produce local topic-plots of teams and developers that can be used to investigate more personally relevant and local trends. This information can help provide feedback to a manager who could augment their project dashboard with these topic-plots in order to observe global trends, personal trends or team-wise trends within these plots. The potential for dashboard use was later confirmed in interviews with a program managers (see Section 5.1.1).

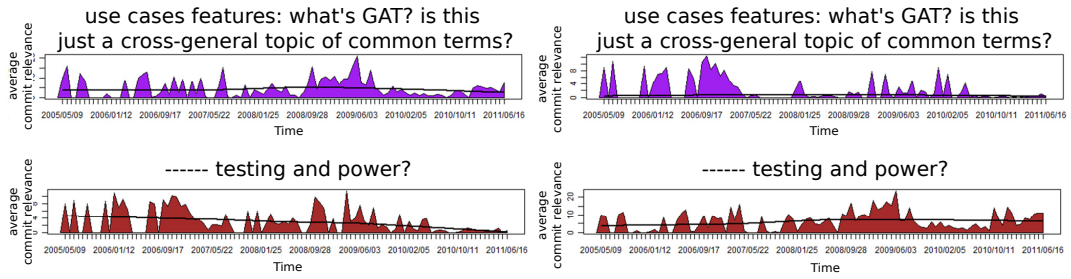
In the next section we discuss the results of the surveys and interviews and the perceptions that developers and managers hold regarding these topic-plots.

5 Qualitative Evaluation

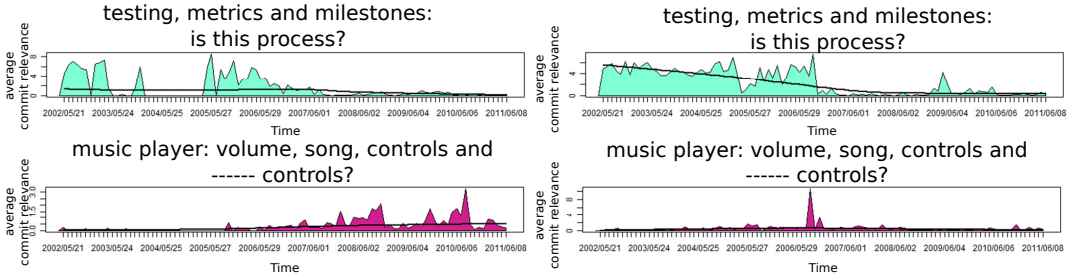
In this section we validate if developers and managers were able to label topics and if the topic-plots matched their perception of the effort that occurred relevant to that topic and its associated requirements.

5.1 Interviews

Initially, we needed to understand requirements use at Microsoft. Thus our initial interviews helped direct the rest of the study. We interviewed one program manager and one developer for one hour each. The interviewees were chosen specifically because they had participated in writing the requirements documents that we had access to. Later during the FLOSS extension of the Microsoft study we interviewed 5 developers over Skype, and IRC. In the following sections we discuss interviews with a program manager and a developer at Microsoft in 2011, and then we summarize the interviews with FLOSS developers in 2013.



(a) topic-plots of 2 developers



(b) topic-plots of 2 teams

Fig. 5: Topic-plots of 2 topics for 2 different developers and teams. The topic labels provided are our non-expert labels. The topic labels shown are non-expert labels.

5.1.1 An Interview with a Program Manager

We interviewed a program manager at Microsoft and asked him to label 3 topics. He walked through each topic word by word and pieced together what he thought the topic was about. Program managers often write requirements and he immediately indicated the relationship of topics to requirements,"I know which specs this came from".

The program manager also indicated that a series of correlated spikes were most likely caused by a *Design Change Request* (DCR), shown in the topics 7 and 9 (first column on the 4th and 5th row of Figure 2). DCRs are management decisions about implementation. They are caused by management wanting a particular change or by external stakeholders, such as vendors, imposing limitations or requirements on certain product features. The particular peak he indicated had to do with bluetooth support.

After the initial topics were labelled the PM voluntarily help label some more topics. When shown a topic-plot (topic 3, 1st column, 2nd row of Figure 2) of a feature that he knew about, the program manager pointed to a dip and mentioned that the feature was shelved at that time only to be revived later, which he illustrated as the commits dipped for a period and then increased. This indicated that his perception of the topic and topic-plot matched reality:

many of the labelled topic-plots mapped to the perception of the program manager. This match between topic and topic-plot is important because the topic-plot might not depict the relevant effort related to the topic or the label given to that topic.

The program manager expressed interest in the research and suggested it would be useful in a project dashboard. He additionally suggested that he would like to be able to "drill down" and see related documents. We then interviewed a developer to see if their view of the project differed.

5.1.2 An Interview with a Developer who Writes Requirements and Design Documents

The Microsoft developer we interviewed had written design documents based on the requirements document. At first he seemed apprehensive about labelling topics, potentially due to the unstructured nature of the task. We had him successfully label the 3 topics that the program manager had labelled. The developer labels correlated (contained similar words and concepts verified by the paper authors) with the program manager as well as our non-expert labels. Some topics were relevant to concepts and features he had worked on. The developer quickly recognized them and explained to us what the topic and those features were actually about.

The developer also mentioned that some topics were more cohesive and less general than others. Since the developer had made commits to the project's source code, we were able to present him with a personal view of his commits across the topics. The developer expressed that this personal view was far more relevant to him than a global view. The developer also agreed that for 2 of the 3 topic-plots, which the developer had labelled, we presented, the plots were accurate and clearly displayed the effort he put into them at different times. When asked about the usefulness of this approach, the developer indicated that it was not useful for himself but might be for managers.

Our preliminary conclusions from the interview were that developers could label topics with some prompting, but were far more comfortable dealing with topics relevant to their own code. The developers preferred topic-plots relevant to themselves over plots about entire project and could pick out and confirm their own behaviour. Whether or not this would be evident in the surveys remained to be seen.

5.2 Survey Responses

In this section we discuss the survey responses for both Microsoft managers and developers, and FLOSS developers.

We administered surveys over email, over IRC, once over Skype, and in person. The surveys at Microsoft occurred during the last Summer and early Fall of 2011, the FLOSS surveys occurred during February and March 2013. In IRC and person surveys were favoured due to low response from the email

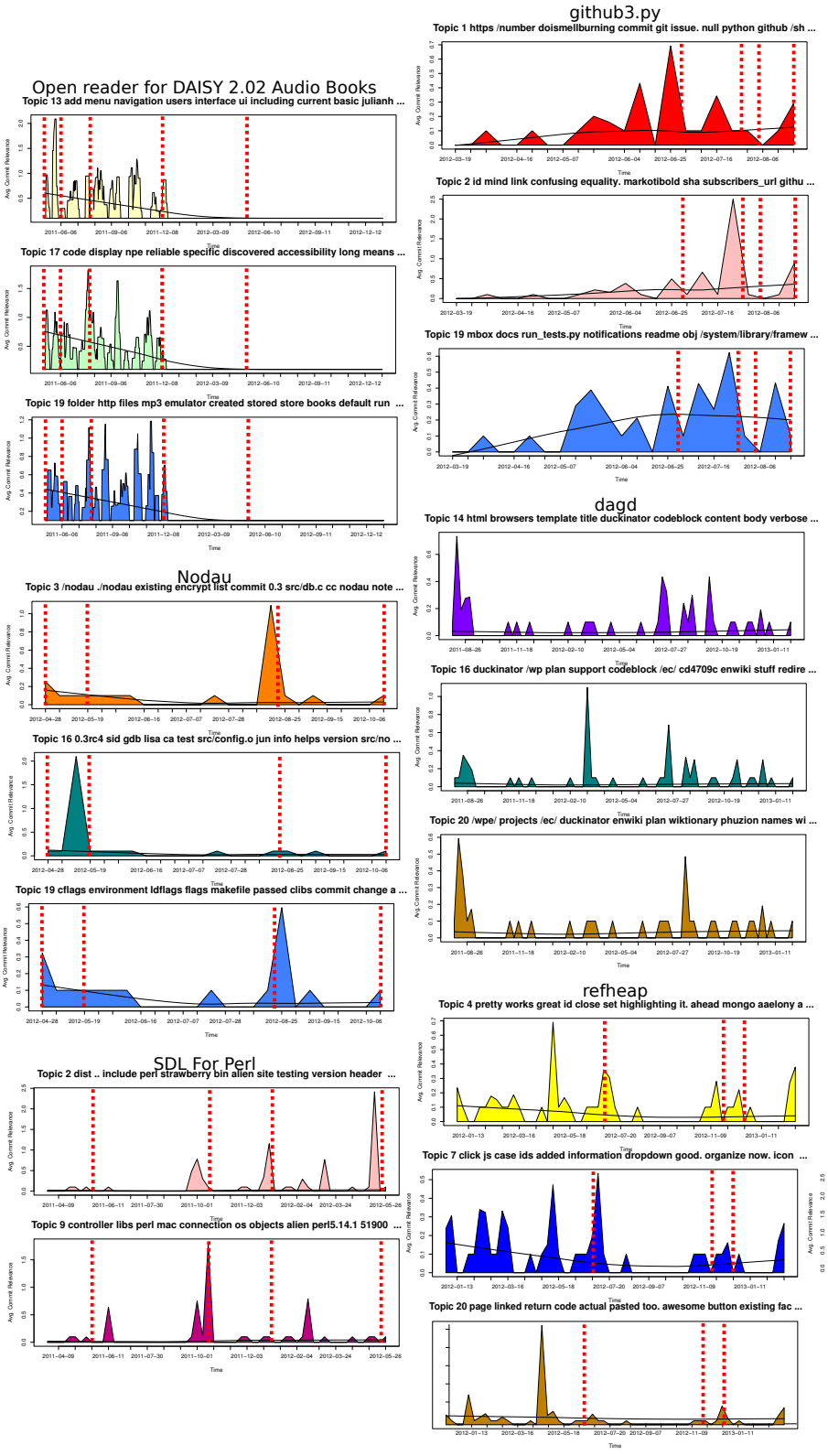


Fig. 6: Issue Report Topic-Plots presented to FLOSS Developers. Each shows the topic-plots that developers were asked to label in the survey in no particular order. See figure 7 for more. Red dashed lines indicate a release or milestone of importance (major or minor). The topic labels shown are the top ranked LDA topic words.

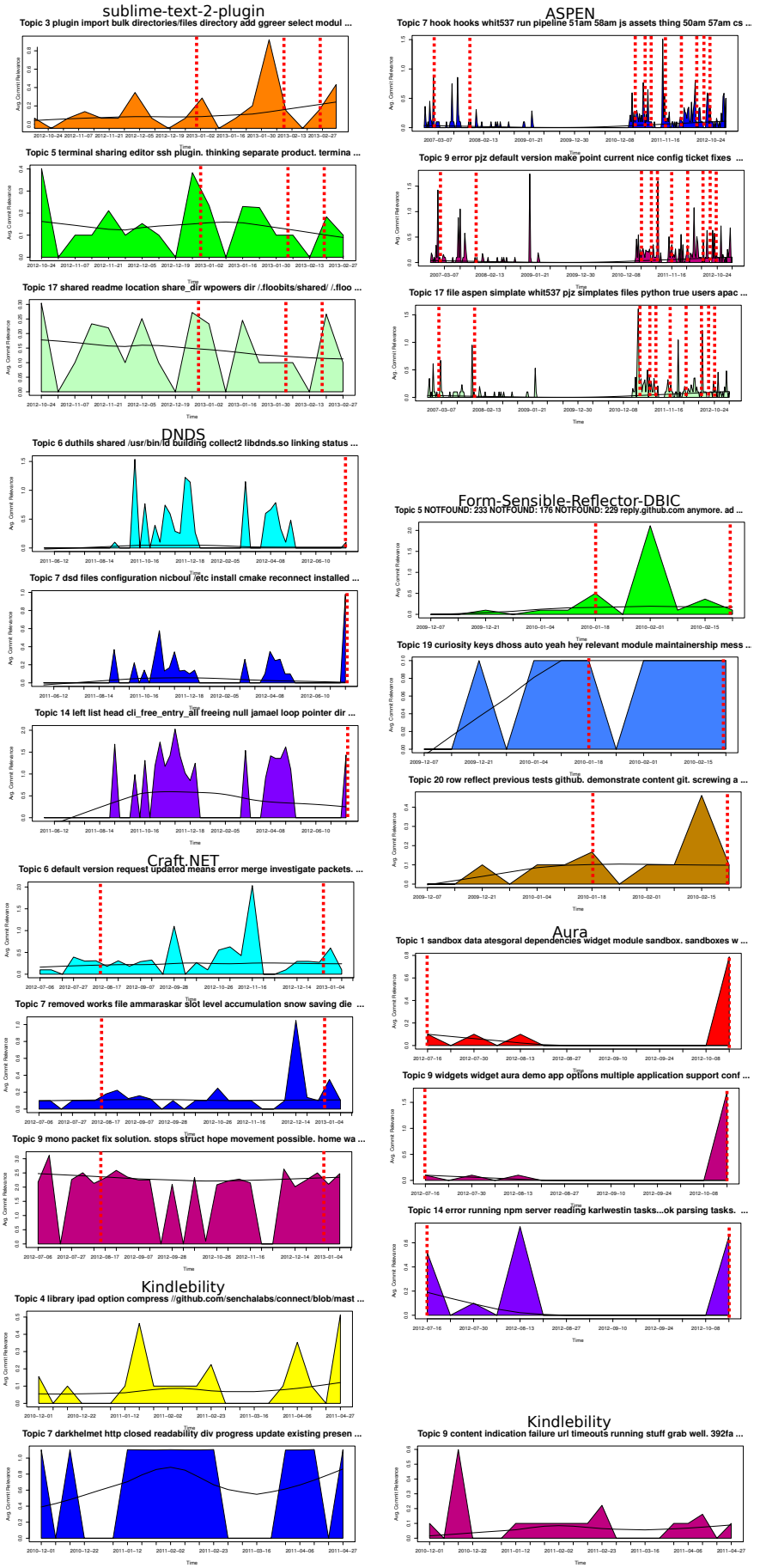


Fig. 7: Issue Report Topic-Plots presented to FLOSS Developers. Each shows the 2 to 3 topic-plots that developers were asked to label in the survey in no particular order. See figure 6 for more. Red dashed lines indicate a release or milestone of importance (major or minor). The topic labels shown are the top ranked LDA topic words.

survey. IRC surveys are surveys where we chatted with the developer and then asked if we could send them an email survey. One developer requested voice administration of the survey, and so we talked with that developer over Skype. One email respondent expressed difficulty interpreting the topic-plots but did associate the behaviour with their own experience:

> Again, all my projects only lasted about 2-3 months — the closest thing that made sense in the topics listed is the USB and video work I did, which was done in June-Aug, possibly coinciding with the last spike.

Our observation from some of the surveys was that some raw, unprocessed LDA topics were far too complicated to be interpreted without the training provided by our example topic that we labelled in front of the respondent. For example one respondent described Topic 6 in Table 2 as:

> These seem pretty random. The words from the topic that actually come close to identifying something in my work area are "device update" and "connection".

As the surveys used personalized plots, such as the plots in Figures 4 and 5a, we gained insight on the perception of the respondent if their plot matched their perceived effort. The respondent of this plot said that some of the plot matched his architectural work that he had concluded that labelling is a difficult activity. The respondent also said that the peaks were in sync with the changes that he had made. In the two other topic-plots he could not recognize any behaviour. Thus some of the topic-plots match his perception, but not all topic-plots were relevant.

FLOSS developers were given personal topics plots as well. All of the personal topic-plots of the FLOSS developer topics are depicted in Figures 6 and 7.

Some respondents found that part of the plots presented to them matched their behaviour while other parts of the same plots did not. "I would have expected more activity," said one Microsoft participant about a topic that was related to client server interactions. FLOSS developers, such as Gerson Goulart and Julian Harty, expressed some doubt:

> I do recognize that I put some small work in this project at the beginning and a bit more on it some time later on, but the dates and plot alone are not enough for me to be confident about how much work was put into each of the topics (which I don't know either from the top of my mind). But a combined plot with different colours labelling each topic could certainly help with that.
>
> – Gerson Goulart regarding AURA topic-plots in Figure 7.

> I recognize my activity varied over the duration of the project; so I would expect peaks and troughs. However I don't know what the graph is based on and I've not compared it to my actual activity on the project.
>
> – Julian Harty regarding Topic 19 of Open Reader in Figure 6.

Other FLOSS developers were more positive: Ricky Elrod of `dagd` said, "I was able to match up some of the spikes in this one. Very neat!" of Topic 14 of dagd in Table 3. Lisa Milne of `nodau` said, "The plot shows the time leading up to a release, it quite clearly shows where a release candidate was pushed out, followed by the actions taken following feedback from users." of Topic 3 of nodau in Figure 6.

The majority of topics were labelled by industrial respondents, and the mode score was 4 for agreement (rather than 5 for strongly agree) that the topic-plot matched the developer's perception. Figure 8 displays the scores for the industrial administered survey: 3 not applicable, 3 strong disagree, 4 disagree, 3 neutral, 10 agree and 1 strongly agree. This gives us a median of 4 (agree) and an average of 3.09 (neutral to agree) from 21 topic ratings. Disagree versus agree, ignoring neutral, had a ratio of 7:11.

The FLOSS developers rated the topic-plots differently: 1 strongly disagree, 8 disagree, 10 neutral, 15 agree, and 3 strongly agree. This is a median of 3 (netural), an average of 3.30 (neutral to agree), and a mode of 4 (agree) from 37 ratings. Floss developers also answered that for 6 of 36 topics that the topics were not relevant to their project or development, while 30/36 were (83%). Disagree versus agree, ignoring neutral had a ratio of 1:2 (9 to 18), and were not statistically different from the industrial respondents (7:11) according to a X^2 test ($p > 0.94$) of *FLOSS Agree* (18) and *FLOSS Disagree* (9) counts versus *Industrial Agree* (11) and *Industrial Disagree* (7). The X^2 test run was the Pearson's Chi-squared test with Yates' continuity correction, resulting in 1 degree of freedom and an X^2 value of 0.004.

"This plot matches my perception of the effort that went into that topic and/or its features," $46-48\%$ of topic-plot ratings were in agreement with this statement for both industrial and FLOSS developers (see Figure 8).

After the surveys were administered to FLOSS developers we interviewed those who had volunteered for a follow up interview.

5.3 Interviews with FLOSS Developers

We interviewed the FLOSS developers as part of this extension to the original study, thus they were interviewed in February and March of 2013, 1.5 years after the initial Microsoft Study. The FLOSS interviews occurred after they had been administered a survey.

Some of the FLOSS developers who answered our survey agreed to be interviewed. We interviewed the FLOSS developers after the Microsoft study, thus we focused more on issues of bug/issue report quality and topic coherency while talking with FLOSS developers.

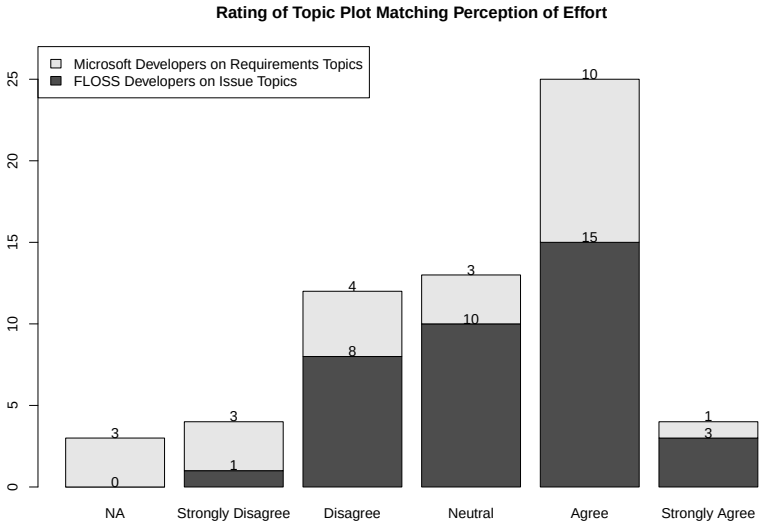


Fig. 8: Distribution of topic-plot perceptual ratings performed by Industrial Participants (light grey) and FLOSS developers (dark grey). On this stacked bar-chart, the top number represents the count of Microsoft Developer ratings, the bottom number is the count of ratings from FLOSS developers. The total height indicates how many total ratings combined.

In total we interviewed 5 out of 13 FLOSS developers. Most interviews with FLOSS developers were executed using textual private messages on the FreeNode IRC Network (4 interviews) and one interview used voice over Skype (1 interview). The interview language used was English although one of the discussions started in French. IRC private messages were beneficial because they were transcribed. The Skype interview was recorded via note-taking.

When we asked FLOSS developers about how they felt labelling topics, Bouliane answered, "I was feeling happy, it's fun to see someone interested in what you're doing. But I felt at the same time confused a bit, by what I was suppose to realize by reading the keywords." Geoffrey Greer said he felt, "mostly confusion with a little bit of amusement". Ian Cordasco said that topics were easy to label, but the quality of the topic suffered due to tokenizing employed, "When listed in plain text separate from the image it was easy. I think the inclusion of punctuation also made it misleading because I unconsciously tried to read it as a sentence." Chad Whitacre pointed out punctuation causing issues in Topic 9 of ASPEN (2nd row, 2nd column of Figure 7), "Here we've got a username again, and a few cases where punctuation doesn't seem to be properly accounted for. The remaining terms don't really call to mind a coherent concept, issue, or feature, however." Tobias Leich said:

> Well, it is easy to spot single words and make connections in [my] mind [about which] problems/features are meant [by the topic]. The hard part was to guess what was meant by them [(the words)] together because the words [themselves] mean so [many] different things.

When asked about the quality of issue reports, FLOSS developers such as Nicolas J. Bouliane lamented the lack of training or templates given to issue reporters. Bouliane suggested that guidelines used by projects such as Asterisk[11] should be helpfully posted so that issue reporters can understand what programmers need.

Some FLOSS developers, such as Bouliane of DNDS, could forsee this kind of work being integrated into Github, or similar tools, in terms of prediction and effort estimation:

> What I see could be nice is something that can evaluate the velocity of task effort put in[to] a task is fun to know, but then is nice when you can relate that [information] with a task you haven't done yet [in order] to approximate how much time it could take you.
>
> I guess checking only the past would be easier at first – this task took you that long , that much effort, that much code, and you need to interfere that much with the existing code.

5.4 Did Respondents Agree on Topic Labels?

We wanted to see if respondents labelling the same topic agreed on the same labels. Only our industrial study had topic label overlap since we never had more than 1 FLOSS developer from 1 project at a time.

In table 2 we can see a selection of industrial topics, their topic words, expert labels and non-expert labels. We can see examples of agreement and disagreement, for instance topic 18 we can see all of the respondents and interviewees agreed that the topic was about setup but whether or not it was a networking device setup or application setup was undetermined. We perceived that familiarity of a developer the requirements documents relevant to the topic aided their ability to label the topic. Topic 15 described in 2 suffered from many experts claiming a lack of coherency, while there was some agreement on the topic of networking. Topic 15 has some agreement with the non-expert, as the redacted term is the same redacted term in the expert labels. This table helps illustrate how agreement exists along a gradient that is subject to subjectivity.

One topic, Topic 19 (depicted in Figure 2) was non-expertly labelled "wifi networks and —— access points", had agreement between 2 of the respondents. One said: "Configuration of [—— access point]", the other said " but really [it is] [—— access point]. [In particular], the [user interface] related to configuring network broadcast, connectivity and sharing."

[11] Asterisk issue tracker guidelines: `https://wiki.asterisk.org/wiki/display/AST/Asterisk+Issue+Guidelines`

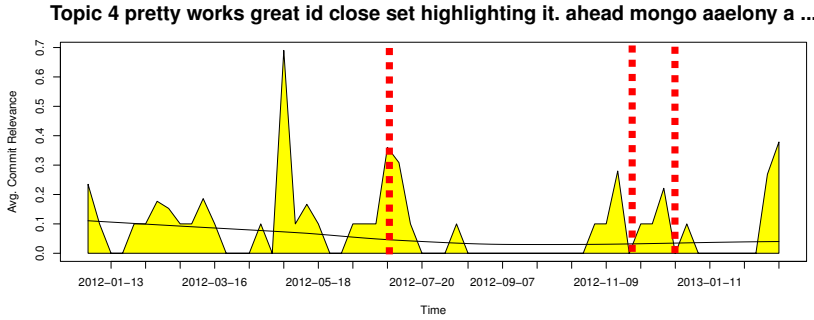


Fig. 9: Topic-plot of Topic 4 from `refheap` with topic words: pretty works great id close set highlighting it. ahead mongo aaelony avoid issue. defined lein2 jquery.hotkeys implement query make site. Red dashed lines indicate a release or milestone of importance (major or minor). The topic label shown consists of the top ranked LDA topic words.

For Topic 5 (in Table 2), two of the respondents had agreed it was a user scenario "End user usage pattern" and "Functionality related network mode ——, allowing uses to select and their preferred network ——". We cannot be sure that either interpretation is more accurate. This illustrates that there can be disagreement in terms of the meaning of a topic.

6 Discussion

6.1 Topic Labelling

We have demonstrated that stakeholders can indeed label topics. Furthermore we have demonstrated that many of these topics are relevant to the task at hand.

6.1.1 Developers

Developers seemed to express difficulty labelling topics, but to be clear, many developers did not write the requirements or the design documents, or the issue reports that the topics were derived from. We argue that some of the difficulty of labelling topics derives from the experience one has with the topic's underlying documents.

There was some expression of irrelevance of some of the topics from the FLOSS developers. Anthony Grimes of `refheap` gave Topic 4, depicted in Figure 9, this label:

> jquery.hotkeys is from when we were giving refheap keyboard shortcuts. Highlighting is probably related to highlighting a line number when you click it. The rest are way too vague to be of use.

Topic 1	Words	0 IC state thread return connection call wininet cm dword api feature component system nat callback query guid dns typedef
	Non-Expert	*wifi code?*
	Expert	"I guess it's about —— and connection sharing" paraphrased
	Expert	Deliverable plan, message plan , delivering messages to —— networks
	Expert	About —— tech and network tech (talk to —— team)
Topic 5	Words	network —— connect wifi hotspot connection probe —— state notification cm time service ui ie authentication reconnect go hotel delete
	Non-Expert	—— *networks? Stuff like hotspots, hotel wifi and* —— —— *network*
	Expert	—— is like hotspot, it's a wifi service.
	Expert	End user usage pattern (SQUM)
	Expert	Functionality related network mode ——, allowing uses to select and their preferred network
	Expert	Wifi services
Topic 6	Words	supl —— node hslp request —— use SETUP certificate registry configuration occurrence mode valid default gat dfproperties shall dftype accesstype
	Non-expert	—— *and SETUP + certificates??*
	Expert	Supl & —— are protocols for ——
	Expert	—— core team in OSPlat Commercialization connectivity (making a ——)
	Expert	"Didn't touch —— code"
	Expert	These seem pretty random. The words from the topic that actually come close to identifying something in my work area are "device update" and "connection"
	Expert	—— related – does not apply to shell code
	Expert	—— and —— support; lack of coherence filled with buzzwords (keywords that managers like).
	Expert	Same domain (networking). ——
	Expert	Update a —— supported on device
Topic 15	Words	test device scenario case —— use —— pass team automation document —— feature work change network ran plan —— ——
	Non-Expert	—— *testing and power?*
	Expert	Not coherent
	Expert	—— Connectivity
	Expert	Not coherent
	Expert	—— networking + generic words
Topic 18	Words	data use file request —— change service new send time fix gat follow —— feature registry value application ——
	Non-Expert	*use cases feature : whats GAT? : is this just a cross-general topic of common terms?*
	Expert	About RIL (radio interface). Relevant to ——. Service plan visibility. GUID for something related to SETUP and ——.
	Expert	SETUP something that changes —— state (over air or internal)
	Expert	Setup application - setup or kind of configuration of an application. SETUP device through setup

Table 2 Topic labelling: the emphasized labels are the non-expert labeling made by the authors. The first list of words is the top 20 words in the topic. The labels are from program managers and developers working on the project (both survey and interviews). —— indicate redactions.

Some FLOSS developers, such as Julian Harty pointed that some topics are indeed irrelevant and hard to label, "The terms you have collected are nonsensical and don't really communicate much about the project at all." Ricky Elrod also noticed that topics, such as Topic 20 shown in Figure 6, might be relevant to the project, but not exactly useful:

> /wpe/ refers to a url route that an issue referenced, but was never used in the project (It was decided against – we used /ec/ instead). "duckinator" and "phuzion" are both usernames of contributors. Relevant to the project, but not to the logic of the project.

Table 3 depicts 1 topic from each participating FLOSS Developer. It also shows the associated discussion or label that the developer associated with that topic.

6.1.2 Managers

Managers seemed to have a higher level view of the project, as they had the awareness that there were other teams and other modules being worked on in parallel. This awareness of the requirements documents and other features, suggested that topic labelling is easier if practitioners are familiar with the concepts. These plots are relevant to managers investigating who have to interpret development trends.

One manager actually gave the labelled topics a scope ranking: low, medium, or high. This scope related to how many teams would be involved, a cross cutting concern might have a high or broad scope, while a single team feature would be have a low scope. This implies that global awareness is more important to a manager than a developer.

6.1.3 Non-Experts

We consider ourselves to be experts in software, but not experts about the actual products that we studied. Thus we relied on the experts to help us label the topics. At the same time we also labelled the topics in the industrial case study. We examined all of the topic labels and checked to see if there was agreement between our labellings and theirs. If our label intersected the semantics or concepts of another label we manually marked it as a match. Of 46 separate industrial labellings (10 labellings from interviews, 12 labellings from email, and 24 labellings from face to face surveys), our labels agreed with the respondents only 23 times.

Only 50% of expertly labelled topic labels were similar or agreed with the non-expert topic labels.

Furthermore at a per-topic level, the average manual agreement between developer expert labels and non-expert labels was agreement with 40% of the

Julian Harty	Open reader for DAISY 2.02 Audio Books	19
folder http files mp3 emulator created stored store books default run developer properly update suggestion 3. copied force directory /		
The terms you have collected are nonsensical and don't really communicate much about the project at all.		

Lisa Milne	nodau	19
cflags environment ldflags flags makefile passed clibs commit change automake make additional e.g. sense darkrose salvatore add carnil variables. version		
A: Appears to be related to the Makefile changes that were [incorporated] after discussions with distribution package maintainers.		

Tobias Leich	SDL for Perl	2
dist .. include perl strawberry bin alien site testing version header installed path copying run script installation module http directory		
This topic is about windows specific issues, like what library version was chosen during installation using Alien::SDL. And I'd say this is about issues where files dont get copied to the right directory.		

Ian Cordasco	github3.py	1
https /number doismellburning commit git issue. null python github /sha fairly events_url merging fork lack object contents_url indent repos_url /key_id		
This seems more like work I performed on the repository section of the API. doismellburning contributed a pull request that changed the behaviour of some repository related methods and the rest of the words seem to agree with this association.		

Ricky Elrod	dagd	14
html browsers template title duckinator codeblock content body verbose pursue handle cc doctype output /head head /body /html simple simple.		
Ah, this is much better. It includes words relevant to the application's purpose. It still includes usernames though. Consider filtering them out?		

Anthony Grimes	refheap	7
click js case ids added information dropdown good. organize now. icon languages tiny migrate started necessarily antares_ questions revisions shortened		
Looks like it is related to when we switched to using the 'chosen' javascript library for the language dropdown. Antares_ switched it to using a different mongodb library called monger. Not sure how those two things are related. That's all I can work out for sure.		

Geoffrey Greer	sublimetext2plugin	5
terminal sharing editor ssh plugin. thinking separate product. terminals. it. nodejs reverse source require plugins belong check fun parts bright		
These seem to be some phrases from commit messages and github issues. Its weird how it focuses on the terminal sharing feature that we decided to drop.		

Nicolas J. Bouliane	DNDS	7
dsd files configuration nicboul /etc install cmake reconnect installed path packages config_file coded /usr/local/etc conf /etc/dnds/dnc.conf hard .deb current define		
– I feel like these words are related with the fact that I moved from autotools to cmake and the effort to make it work. It also reflect the fact that I did some cleanup in the way I was linking my libraries.		

Drew DeVault	Craft.NET	7
removed works file ammaraskar slot level accumulation snow saving die errors strwarrior effort additional jump detection reduction savedlevel worth organization		
ammaraskar strwarrior devs phases of developers saving levels to disc refactoring levels weather managment (accumulation snow) jump detection		

Daniel Huckstep	kindlebility	4
library ipad option compress //github.com/senchalabs/connect/blob/master/lib/connect/middleware/gzip.js gzip random catching invalid worth protocol error throws kind request darkhelmet 8bdfa63cb871bd6c557a589aefa7935328eda812 functions. 8bdfa63cb871bd6c557a589aefa7935328eda812. Https		
It's a little all over the place, but it's mainly about a third part library I used. It's about the http framework I used.		

Chad Whitacre	Aspen	17
file aspen simplate whit537 pjz simplates files python true users apache work main module create web line works virtual import		
These are all words that more or less relate to Aspen, in terms of general concepts: Aspen is a python web framework that uses the file system extensively and specifically a file pattern called simplates, and whit537 and pjz are the two main authors. None of these really evoke specific issues or features, however.		

Devin Joel Austin	Form-Sensible-Reflector-DBIC	19
curiosity keys dhoss auto yeah hey relevant module maintainership messages taking things 2011 impact email roles added anymore guess would.		
email roles added anymore guess would.		

Gerson Goulart	Aura	1
sandbox data atesgoral dependencies widget module sandbox. sandboxes widgets method data. fetch core individual instance understand root collections list looked		
At one point of the project I found another project on Github created by @atesgoral that had better concept of sandboxes. I created an entry in Aura about it and @atesgoral himself jumped in to share ideas.		

Table 3 One Issue Tracker Topic per Project/FLOSS Developer associated with a Label by the FLOSS Developer. The first row consists of the Developer's name, the project and the topic number (the identity of the topic) they labelled.

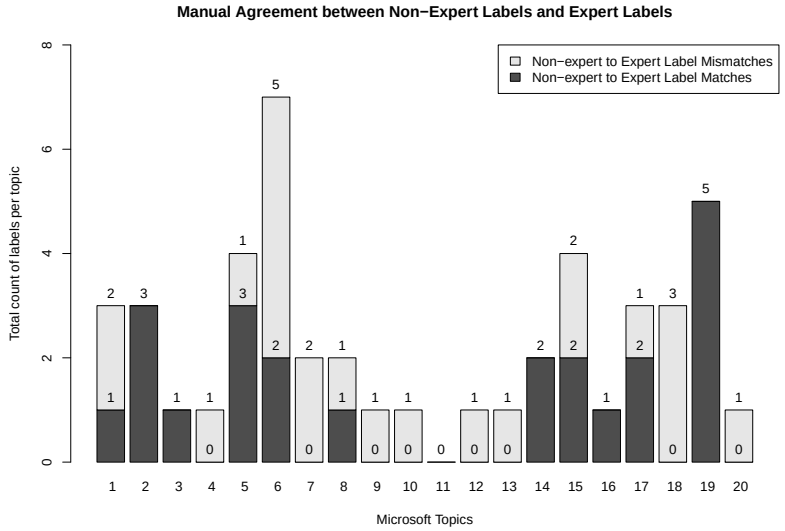


Fig. 10: Agreement between Expert Labels and Non-Expert Labels per each Industrial Requirement Topic. On this stacked bar-chart, the numbers on top of dark grey bars indicate the number of matches (agreement in labels), the numbers on top of light grey bars indicate the number of mismatches (disagreement in labels).

labels (the mean of agreement divided by total per topic). See Figure 10 for a chart of non-expert versus expert labels in terms of agreement. This indicates that while non-experts can label these topics, there is inherent value in getting domain experts to label the topics.

6.2 FLOSS Developers

All of the FLOSS developers who answered the survey required that their contribution be attributed to them. Perhaps this is because most FLOSS licenses, including the permissive BSD and MIT licenses, all require attribution as one of the main requirements of the license. Some of the motivation for participation might have been to promote their project to a wider community.

6.2.1 Did FLOSS Developers differ from Industrial Participants?

In order to address the question, "Did FLOSS Developers differ from Industrial Participants?" we compared the distributions of the two studies shown in Figure 8 using the X^2 test (Pearson's Chi-Squared Test). We used both immediate and simulated tests X^2, and we included and excluded *Not Applicable* responses, but all our tests found that the FLOSS Developers and Industrial

Developers perceptual distributions were not statistically significantly different. Our X^2 tests had p-values between 0.12 and 0.44 described in the next paragraph.

For all of the ratings including "not applicable" when we compare FLOSS ratings and industrial ratings using Pearson's Chi-squared test with 5 degrees of freedom, an X^2 value of 8.7285 and resulting $p > 0.12$, which indicates the ratings were similar between groups of developers. By excluding "not applicable" ratings (because it has 0 ratings for FLOSS developers) using Pearson's Chi-squared test with 4 degrees of freedom, with an X^2 of 3.9926 and a resulting $p > 0.40$, we conclude that the ratings were still similar between groups of developers when "not applicable" ratings were excluded.

Thus the ratings of agreement between FLOSS and industrial developers are similar. With more respondents we might achieve more clarity if there is a difference. Thus we conclude we have no statistical evidence that the FLOSS developers responded differently than the Industrial participants.

6.3 Commit-Topic-Plot

We found that the per author commit-topic-plots were often relevant to the developers. Some higher level topics seemed out of touch with the developer's activity. If a plot did not match a developer's perception, we perceived that it could be due to a lack of familiarity with the topic. Perhaps the topic was unclear rather than the plot missing efforts that they expected. We worry in some cases that developers might be matching noise to noise. Many industrial respondents indicated that we should talk to the team that produced the requirements we used. Some FLOSS respondents suggested they did not have much input into the project and were not the most relevant people to speak to. This lends further evidence that it is easier for those familiar with the source requirements to label topics than those who are not as closely associated. Since respondents validated the majority of the plots as accurate, this provides evidence that the results are not spurious.

Many FLOSS developers reported issues with the distance between commits or the irrelevance of the topics to their actual work.

6.4 Why did the Administered and IRC initiated surveys work?

The improved response to in person and verbally administered surveys provides evidence to support that the examples we provided and the ability to discuss issues and address uncertainty with the respondents enabled them to better understand what topics were and what labelling topics entailed. Earlier respondents might have been worried about labelling a topic incorrectly, when in fact we sincerely did not know what the topic was about.

The strategy of presenting a FLOSS developer with the global topic plot of their project could have been interpreted as gift, thus making the receiving developer more amenable to give back in-kind.

6.5 Issues versus Requirements

One of the problems with our FLOSS replication of the industrial study was that we lacked requirements documents and instead relied on issue tracker issues. Issues in some FLOSS projects are potentially the closest documents to requirements documents. Sometimes features and even user-stories are described in issues. Unfortunately the use of issues is not uniform and the categorization of issues is lacking at best. Thus the issue tracker will usually contain far more bug reports about the software than feature-requests or requirements. This means that our analysis of topics of issues takes on a software quality/software maintenance perspective that was not apparent in the industrial requirements documents.

Furthermore the size of each issue was far smaller than any of the requirements documents and the number of issues per project often exceeded the number of requirements documents we used. Other differences included how we represented the issues to LDA: we explicitly provided authorship information as that was part of the structure of an issue, and part of the structure of the requirements document text. Yet the authorship information for issues is not embedded in the subject or description of the issue, thus to simulate requirements we prefixed the issue author to the issue documents we fed into LDA.

The language and structure of the requirements documents was dictated by a requirements document template. Issues have no such template and are often free-form save for some categorical features such as severity or module affected. This difference could cause LDA to produce template-related topics for the requirements documents.

7 Recommendations on the Use of Topic Analysis in Software Engineering

Based on our experience, the discussions we had with respondents and the results of our surveys we have compiled general recommendations for the use of topic analysis techniques in software engineering.

Many found that labelling a set of *personally relevant topics are easier to interpret*. Respondents found that topics about familiar artifacts tended to be easier to label. One should use the most specific domain experts to label topics. For optimal results, the team responsible for the requirements should label those topics.

Remove confusing, irrelevant and duplicated topics. Some topics do not actually say anything. Some are structural and filled with common terms, some are about the use of language itself and not relevant to requirements. Most importantly, not all topics need to be shown.

Use domain experts to label topics! We found that non-experts have questionable labelling accuracy (only 50%, with a confidence interval of 35% - 65%). Respondents with the most familiarity gave the most relevant topic labels.

Unlabelled topics are not enough! It took respondents 1 to 4 minutes to interpret a topic from its top topic words. Thus multiple topics multiply the cost of interpretation.

Tokenization matters! Depending on the source text, how tokens are kept or not matter. Splitting on punctuation naively can harm accented words and hamper the interpretation of a topic.

Relationships are safer than content! The relationships between documents and topics extracted by LDA are much safer to rely upon than the content of the topic. The content of the topic can be interpreted many different ways and LDA does not look for the same patterns that people do. Focusing on relationships between topics and documents avoids errors in topic interpretation and attribution.

SE researchers should be careful about interpreting topics! Repeatedly in this study we found that small innocuous words and acronyms often had important project specific meanings that were only clarified by the developers themselves.

Topics linked to effort can provide some form of overview! Based on the results of the original study and its replication we feel confident that topics can be leveraged for the purposes of overview, summary, and dashboard visualization.

8 Threats to Validity

Relevant *construct validity* threats include the fact we used only one large project and 13 smaller projects and that personal topic-plots are relevant only to a single person. We were able to partially mitigate this thread by evaluating with multiple people and multiple FLOSS projects. However, the largest threat facing the construct validity of this work is that we did not have enough respondents. Thus we need to rely on qualitative evidence. Our surveys showed topics in a random order to avoid order bias. Training and verbal administration of surveys can also bias results. Although we administered the survey from a script, the fact that we did so verbally and answered questions about our methodology could introduce bias. Showing FLOSS developers a preview of their project in the IRC channel could have biased their results. Commits evaluated were not filtered if they had a small number of tokens which could lead to low quality topics. Furthermore we rely on LDA topic relevance to associate commits with topics and thus assign effort to topics: construct validity is potentially weakened by the use of commits as a proxy for effort.

In terms of *internal validity*, we built explanations and theories based on the feedback we received from respondents. Since we lacked a large number of respondents we were not able to do statistical analysis, but Ko et al. have argued that this size of result is still relevant [17] qualitatively, as we observed repeated answers. Some inconsistency could arise from our use of two different LDA implementations, a CVB0 implementation at Microsoft and the FLOSS Vowpal Wabbit, but both methods use a variational Bayes LDA implementa-

tion. LDA has many derivations since it is a probabilistic technique. In our FLOSS replication we did not apply stemming as we had in our Microsoft study.

External validity is threatened by the fact that requirements study of this study took place on one project, within one organization. We could not find an alternative project that was publicly available that had enough requirements and maturity. Thus we had to replicate using issue reports due to a general lack of formal requirements documentation internal to FLOSS projects (some exists but we would also need willing participants from those projects). External validity was harmed by failing to replicate the utility questions on our FLOSS developer survey that we used on our Microsoft developer survey.

9 Future Work

Future work relevant to this study includes further validation by expanding the scope in terms of software domains, developers, managers, projects and organizations.

The survey respondents had many great ideas. One respondent desired a UI to dive deep into the relevant artifacts to explain behaviour. Others suggested that providing your own word distribution as a topic would help exploration. One PM suggested that Figure 2 would be useful as a project dashboard. Thus this work can be leveraged in research relevant to knowledge management, project dashboards, project effort models and software quality models.

We would like to investigate the effectiveness of automatic topic labels versus those labels given by developers using methods such as those suggested by Kuhn et al. [21] and De Lucia et al. [9]. The intersection of automatic topic labelling and manual topic labelling could help evaluate automatic topic label quality.

10 Conclusions

In conclusion, we conducted an evaluation of the commonly used practice of LDA topic analysis for traceability research (at a high-level) with Microsoft developers, rather than students, in a large project with comprehensive requirements documents. We also replicated the Microsoft case study on 13 FLOSS developers from 13 FLOSS projects with similar conclusions.

We investigated the relevance of topics extracted from requirements to development effort by interviewing developers and managers. To relate requirements and development activities, we extracted topics from requirements documents using LDA, and then inferred the relationship to the version control commit messages.

We combined a large corpus of requirements documents with the version control system and had stakeholders validate if these topics were relevant and

if the extracted behaviours were accurate. We also confirmed the accuracy of extracted behaviours from issue tracker extracted topics with FLOSS developers. Many topics extracted from requirements and issue reports were relevant to features and development effort. Stakeholders who were familiar with the underlying requirements documents or issues tended to be comfortable labelling the topics and identifying behaviour, but those who were not, showed some resistance to the task of topic labelling. Topics labelled by non-experts tended to be inaccurate compared with expert labels.

Stakeholders indicated that many of the commit-topic plots were perceptually valid. The efforts depicted often met with their expectation or experiences. Managers could spot trends in the global plots while developers tended to spot trends in their personal topic-plots. We found evidence that topics and their relevant commits often match the practitioner's perception of their own effort relevant to a topic. But we also found that some topics were confusing and not easy for practitioners to interpret and label. Our recommendations were that topics need to be interpreted, pruned, and labelled by experts; thus future topic-related research should use labelled topics.

We have shown that topics extracted from requirements are relevant, that their version control inferred behaviour is perceptually valid. In short, we have provided evidence that validates some of the assumptions that researchers had previously made about LDA derived topics and have shown that practitioners can interpret and label topics.

Acknowledgments

Thanks to the many managers and developers at Microsoft who volunteered their time to participate in our research and provide their valuable insights and feedback. Abram Hindle performed some of this work as a visiting researcher at Microsoft Research. Thanks to the Natural Sciences and Engineering Research Council of Canada for partially funding this work. Thanks to Abram Hindle's first student, Zhang Chenlei, for his feedback. Thanks to the FLOSS developers who chose to participate: Julian Harty, Lisa Milne, Tobias Leich, Ian Cordasco, Ricky Elrod, Anthony Grimes, Geoffrey Greer, Nicolas J. Bouliane, Drew DeVault, Daniel Huckstep, Chad Whitacre, Devin Joel Austin, and Gerson Goulart.

References

1. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. Software Engineering, IEEE Transactions on **28**(10), 970–983 (2002)
2. Asuncion, A., Welling, M., Smyth, P., Teh, Y.W.: On smoothing and inference for topic models. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, pp. 27–34. AUAI Press (2009)
3. Asuncion, H.U., Asuncion, A.U., Taylor, R.N.: Software traceability with topic modeling. In: Proceedings of the 32nd ACM/IEEE International Conference on Software

Engineering - Volume 1, ICSE '10, pp. 95–104. ACM, New York, NY, USA (2010). DOI 10.1145/1806799.1806817
4. Baldi, P.F., Lopes, C.V., Linstead, E.J., Bajracharya, S.K.: A theory of aspects as latent topics. In: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, OOPSLA '08, pp. 543–562. ACM, New York, NY, USA (2008). DOI 10.1145/1449764.1449807
5. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
6. Capiluppi, A., Izquierdo-Cortázar, D.: Effort estimation of floss projects: a study of the linux kernel. Empirical Software Engineering **18**(1), 60–88 (2013)
7. Cheng, B.H.C., Atlee, J.M.: Research directions in requirements engineering. In: 2007 Future of Software Engineering, FOSE '07, pp. 285–303. IEEE Computer Society, Washington, DC, USA (2007). DOI 10.1109/FOSE.2007.17
8. Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E., Christina, S.: Goal-centric traceability for managing non-functional requirements. In: Proceedings of the 27th international conference on Software engineering, ICSE '05, pp. 362–371. ACM, New York, NY, USA (2005). DOI 10.1145/1062455.1062525
9. De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., Panichella, S.: Using ir methods for labeling source code artifacts: Is it worthwhile? In: Program Comprehension (ICPC), 2012 IEEE 20th International Conference on, pp. 193–202. IEEE (2012)
10. De Lucia, A., Marcus, A., Oliveto, R., Poshyvanyk, D.: Information retrieval methods for automated traceability recovery. In: Software and Systems Traceability, pp. 71–98. Springer (2012)
11. Ernst, N., Mylopoulos, J.: On the perception of software quality requirements during the project lifecycle. In: R. Wieringa, A. Persson (eds.) Requirements Engineering: Foundation for Software Quality, *Lecture Notes in Computer Science*, vol. 6182, pp. 143–157. Springer Berlin / Heidelberg (2010)
12. Gethers, M., Oliveto, R., Poshyvanyk, D., Lucia, A.D.: On integrating orthogonal information retrieval methods to improve traceability recovery. In: Software Maintenance (ICSM), 2011 27th IEEE International Conference on, pp. 133–142. IEEE (2011)
13. Grant, S., Cordy, J.R.: Estimating the optimal number of latent concepts in source code analysis. In: Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation, SCAM '10, pp. 65–74. IEEE Computer Society, Washington, DC, USA (2010)
14. Hindle, A., Bird, C., Zimmermann, T., Nagappan, N.: Relating requirements to implementation via topic analysis: Do topics extracted from requirements make sense to managers and developers? In: Proceedings of the 28th IEEE International Conference on Software Maintenance. IEEE (2012)
15. Hindle, A., Ernst, N.A., Godfrey, M.W., Mylopoulos, J.: Automated topic naming to support cross-project analysis of software maintenance activities. In: Proceedings of the 8th Working Conference on Mining Software Repositories, MSR '11, pp. 163–172. ACM, New York, NY, USA (2011). DOI 10.1145/1985441.1985466
16. Hoffman, M., Bach, F.R., Blei, D.M.: Online learning for latent dirichlet allocation. In: advances in neural information processing systems, pp. 856–864 (2010)
17. Ko, A.J., DeLine, R., Venolia, G.: Information needs in collocated software development teams. In: Proceedings of the 29th international conference on Software Engineering, ICSE '07, pp. 344–353. IEEE Computer Society, Washington, DC, USA (2007). DOI 10.1109/ICSE.2007.45
18. Koch, S.: Effort modeling and programmer participation in open source software projects. Information Economics and Policy **20**(4), 345–355 (2008)
19. Konrad, S., Cheng, B.: Automated analysis of natural language properties for uml models. In: J.M. Bruel (ed.) Satellite Events at the MoDELS 2005 Conference, *Lecture Notes in Computer Science*, vol. 3844, pp. 48–57. Springer Berlin / Heidelberg (2006)
20. Kozlenkov, A., Zisman, A.: Are their design specifications consistent with our requirements? In: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering, RE '02, pp. 145–156. IEEE Computer Society, Washington, DC, USA (2002)
21. Kuhn, A., Ducasse, S., Gírba, T.: Semantic clustering: Identifying topics in source code. Information and Software Technology **49**(3), 230–243 (2007)

22. Lukins, S.K., Kraft, N.A., Etzkorn, L.H.: Source code retrieval for bug localization using latent dirichlet allocation. In: Proceedings of the 2008 15th Working Conference on Reverse Engineering, WCRE '08, pp. 155–164. IEEE Computer Society, Washington, DC, USA (2008). DOI 10.1109/WCRE.2008.33
23. Marcus, A., Maletic, J.I.: Recovering documentation-to-source-code traceability links using latent semantic indexing. In: Software Engineering, 2003. Proceedings. 25th International Conference on, pp. 125–135. IEEE (2003)
24. Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J.I.: An information retrieval approach to concept location in source code. In: Proceedings of the 11th Working Conference on Reverse Engineering, WCRE '04, pp. 214–223. IEEE Computer Society, Washington, DC, USA (2004)
25. McMillan, C., Poshyvanyk, D., Revelle, M.: Combining textual and structural analysis of software artifacts for traceability link recovery. In: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, TEFSE '09, pp. 41–48. IEEE Computer Society, Washington, DC, USA (2009). DOI 10.1109/TEFSE.2009.5069582
26. Murphy, G.C., Notkin, D., Sullivan, K.J.: Software reflexion models: Bridging the gap between design and implementation. IEEE Trans. Softw. Eng. **27**(4), 364–380 (2001). DOI 10.1109/32.917525
27. Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., De Lucia, A.: How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 522–531. IEEE Press (2013)
28. Poshyvanyk, D.: Using information retrieval to support software maintenance tasks. Ph.D. thesis, Wayne State University, Detroit, MI, USA (2008)
29. Ramage, D., Dumais, S.T., Liebling, D.J.: Characterizing microblogs with topic models. In: ICWSM (2010)
30. Ramesh, B.: Factors influencing requirements traceability practice. Commun. ACM **41**(12), 37–44 (1998). DOI 10.1145/290133.290147
31. Reiss, S.P.: Incremental maintenance of software artifacts. IEEE Trans. Softw. Eng. **32**(9), 682–697 (2006). DOI 10.1109/TSE.2006.91
32. Sabetzadeh, M., Easterbrook, S.: Traceability in viewpoint merging: a model management perspective. In: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering, TEFSE '05, pp. 44–49. ACM, New York, NY, USA (2005). DOI 10.1145/1107656.1107667
33. Savage, T., Dit, B., Gethers, M., Poshyvanyk, D.: Topicxp: Exploring topics in source code using latent dirichlet allocation. In: Proceedings of the 2010 IEEE International Conference on Software Maintenance, ICSM '10, pp. 1–6. IEEE Computer Society, Washington, DC, USA (2010). DOI 10.1109/ICSM.2010.5609654
34. Shull, F., Singer, J., Sjberg, D.I.K.: Guide to Advanced Empirical Software Engineering, 1st edn. Springer Publishing Company, Incorporated (2010)
35. Sneed, H.M.: Testing against natural language requirements. In: Proceedings of the Seventh International Conference on Quality Software, QSIC '07, pp. 380–387. IEEE Computer Society, Washington, DC, USA (2007)
36. Thomas, S.W., Adams, B., Hassan, A.E., Blostein, D.: Validating the use of topic models for software evolution. In: Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation, SCAM '10, pp. 55–64. IEEE Computer Society, Washington, DC, USA (2010). DOI 10.1109/SCAM.2010.13
37. Thomas, S.W., Adams, B., Hassan, A.E., Blostein, D.: Modeling the evolution of topics in source code histories. In: Proceedings of the 8th Working Conference on Mining Software Repositories, MSR '11, pp. 173–182. ACM, New York, NY, USA (2011). DOI 10.1145/1985441.1985467
38. Tillmann, N., Chen, F., Schulte, W.: Discovering likely method specifications. In: Z. Liu, J. He (eds.) Formal Methods and Software Engineering, *Lecture Notes in Computer Science*, vol. 4260, pp. 717–736. Springer Berlin / Heidelberg (2006)
39. Wiegers, K.E.: Software Requirements, 2 edn. Microsoft Press, Redmond, WA, USA (2003)

40. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering: an introduction. Kluwer Academic Publishers, Norwell, MA, USA (2000)