Relating Requirements to Implementation via Topic Analysis: Do Topics Extracted from Requirements Make Sense to Managers and Developers? Web Edition

Abram Hindle Department of Computing Science
University of Alberta
Edmonton, Canada
Email: abram.hindle@softwareprocess.es
Christian Bird and Thomas Zimmermann and Nachiappan Nagappan Microsoft Research
Redmond, WA, USA
Email: {cbird,tzimmer,nachin}@microsoft.com

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 in the version control system, using Latent Dirichlet Allocation (LDA). We evaluate LDA topics on practitioners and check if the topics and trends extracted matches the perception that Program Managers and Developers have about the effort put into addressing certain topics. We found that effort extracted from version control that was relevant to a topic often matched the perception of the managers and developers of what occurred at the 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 to version control commits via topic analysis and validate with the actual stakeholders the relevance of these topics extracted from requirements.

Keywords

latent Dirichlet allocation (LDA); requirements; version control; traceability; topics; requirements engineering

I. Introduction

For many organizations, requirements and specifications provide the foundation for the products that they produce. As requirements are implemented, the link between requirements and implementation weaken, especially during maintenance. Later, development artifacts often stop referencing the requirements documents that they were based on. Current research shows that there is a lack of traceability between requirements and implementation [1] whereas the managers we interviewed expected and wanted requirements and implementation to be in sync (Section V-B). The volume of traceability research confirms its importance (see, for example, [2]–[5]). In this paper we extract topics, word distributions, from a large body of requirements documents and then search for commits that mention these topics within the version control system. Once labelled, these topics provide some high-level traceability between requirements and implementation, yet at the same time they can provide an overview of development effort for each topic.

Furthermore we attempt to validate these overviews between requirements and implementation by asking developers and program managers if their perception of their behaviour matches the behaviour highlighted by the topic. This is a validation of if LDA topics and topics extracted from requirements and related to commits make sense to practitioners. Stakeholder based validation of topics (extracted by Latent Dirichlet Allocation (LDA) [6]) in terms of relevance, labelling and the recovery of behaviour [7] is critical, but to date has not been applied to the domain of software engineering. 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 developer and managers.
- An analysis of whether topic highlighted behaviour matches the perception of developers and managers.
- 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.

We are investigating if LDA topics make sense to practitioners, and can they label LDA topics. For this study, we have at our disposal many skilled Microsoft developers who work on a very large software system that has many requirements documents. 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.

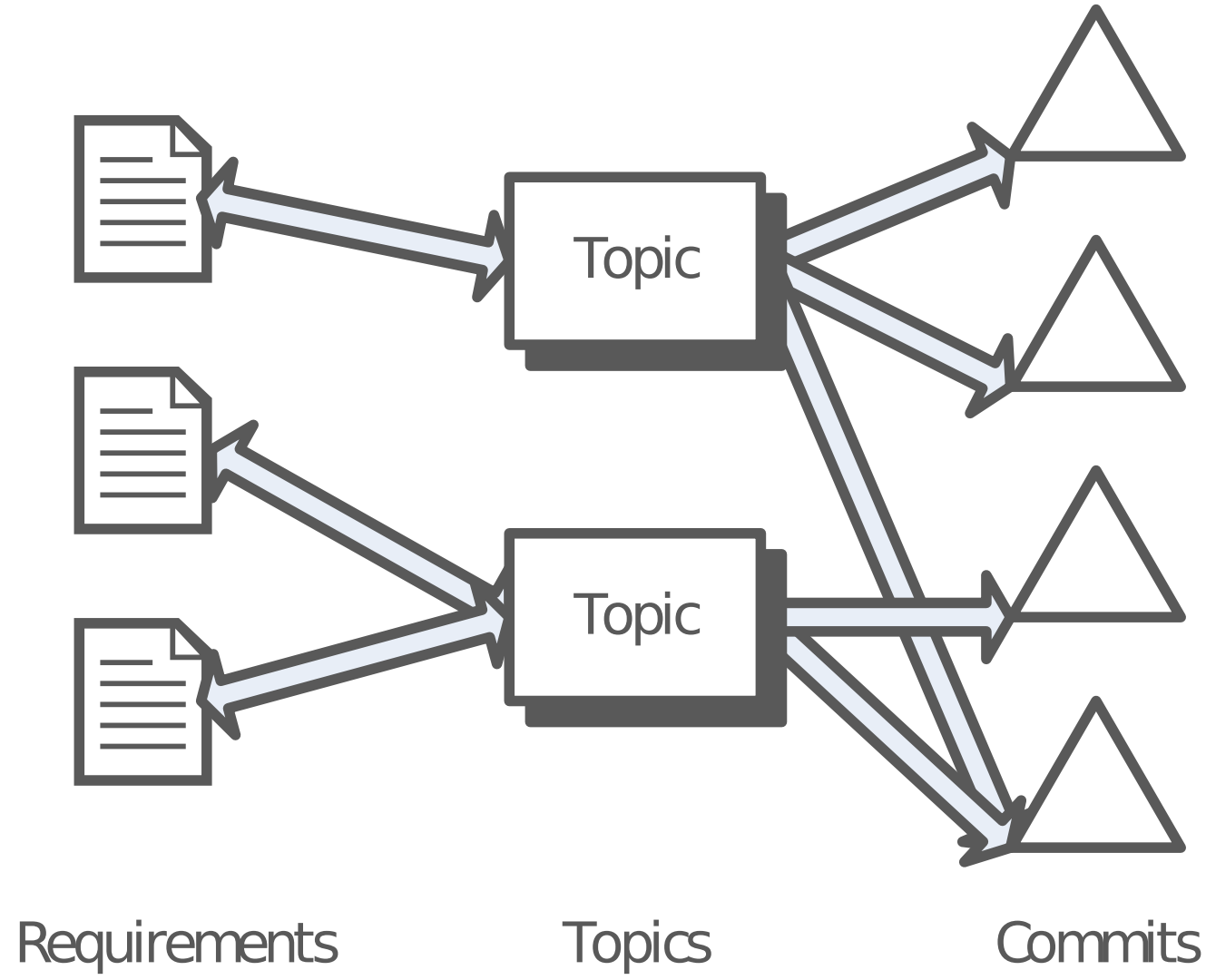


Fig. 1. Our data model, Topics are extracted from Requirements using LDA. Topics are related to requirements in a many-to-many relationship. Commits are related to topics via inference.

A. Motivation

Latent Dirichlet Allocation (LDA) and Latent Semantic Indexing (LSI) and are popular tools in software engineering research [8]–[11] used for traceability and information retrieval, but their use is built upon assumptions, such as usability, that have not been validated with stakeholders, such as developers. Often these topics are interpreted solely by the researchers themselves (e.g., [12]). Thus the use of topic analysis, LDA and LSI, and topics in software engineering has not been validated rigorously 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.

First and foremost, we believe named topics allow stakeholders such as managers to track development efforts related to these extracted topics. These topic plots allow for local analysis in order study requirements-relevant effort of different groups of developers (teams). One example concrete scenario is a manager identifying how much effort and when different aspects of the requirement documents were addressed. Managers could try to answer the question like Who should I talk to regarding a requirement change in the bluetooth stack? using individual developer topic plots.

II. Background and Previous Work

Our work fits into traceability, requirements engineering, and topic analysis [4], [5], [11], [13].

A. Traceability and Requirements Engineering

Traceability in software engineering is a well-studied topic and authors such as Ramesh et al. [14] have provided excellent surveys of the literature and the techniques. Karl Wiegers [2] has argued about tagging commits with relevant requirements to

aide traceability. Gannod et al. [3] have discussed mining specifications by their language in order to aid traceability for reverse engineering. Kozlenkov and Zisman et al. [1] have also studied requirements traceability with respect to design documents and models. Ernst et al. [15] attempted to trace nonfunctional requirements (NFRs) within version control using a simple technique incorporating NFR word dictionaries. Murphy et al. [16] defined explicit mappings between concepts and changes to files but did not use topics, whereas Sneed [17] investigated mining requirements in order to produce test cases. Reiss et al. [18] produced a tool, CLIME, that allowed one to define constraints and track the co-evolution of artifacts in order to enforce adherence. Poshyvanyk et al. [9], [19], [20] 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) to produce UML models [21]. NLP is also used in topic analysis.

B. 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) [6] 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 linear combination of the n topics that are extracted. Thus the end result of topic analysis is a set of n topics (word distributions) and a matrix that provides the relationship between documents and topics. This means that 1 document can be related to 0 or 1 to n topics. Since each LDA topic is a word distribution over many words, we must present to end-users (developers and managers) an alternative representation. These topics can be represented to end-users as a ranked list of words, from highest magnitude to lowest magnitude. Many researchers use top-10 lists of words, in this study we used 20 words. An example topic might be:

code improve change functionality behaviour readability maintainability structure restructure modify reduce quality process complexity software re-factoring performance

How would you label this topic? Notice how this topic takes time to interpret. 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 [12] to Latent Semantic Indexing (LSI) [22]. Researchers such as Poshyvanyk et al. [19] and Marcus et al. [22] often focus on the document relationships rather than the topic words. In terms of topics and traceability, Baldi et al. [8] labelled topics and then tried to relate topics to aspects in software. Asuncion et al. [7] used LDA on documentation and source code in order to provide traceability links. Grant et al. [23] have worked with LDA and topics before, their 2010 paper suggests heuristics for determining the optimal number of topics to extract. Thomas et al. [11] statistically validated email to source code traceability using LDA. Both groups did not validate their results with practitioners.

These studies rely on assumptions about the applicability to practitioners. We investigate 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 controls 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.

III. Methodology

Our goal is to relate commits to topics of requirements and then see if these topics and plots of topic-relevant commits make sense to stakeholders such as developers and managers. Our methodology is to extract requirements, 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.

A. 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 1500 pages of documentation, 59000 lines of text, 35000 paragraphs, and 285000 words. The average document was 20 pages long with an average word count of 3800 words.

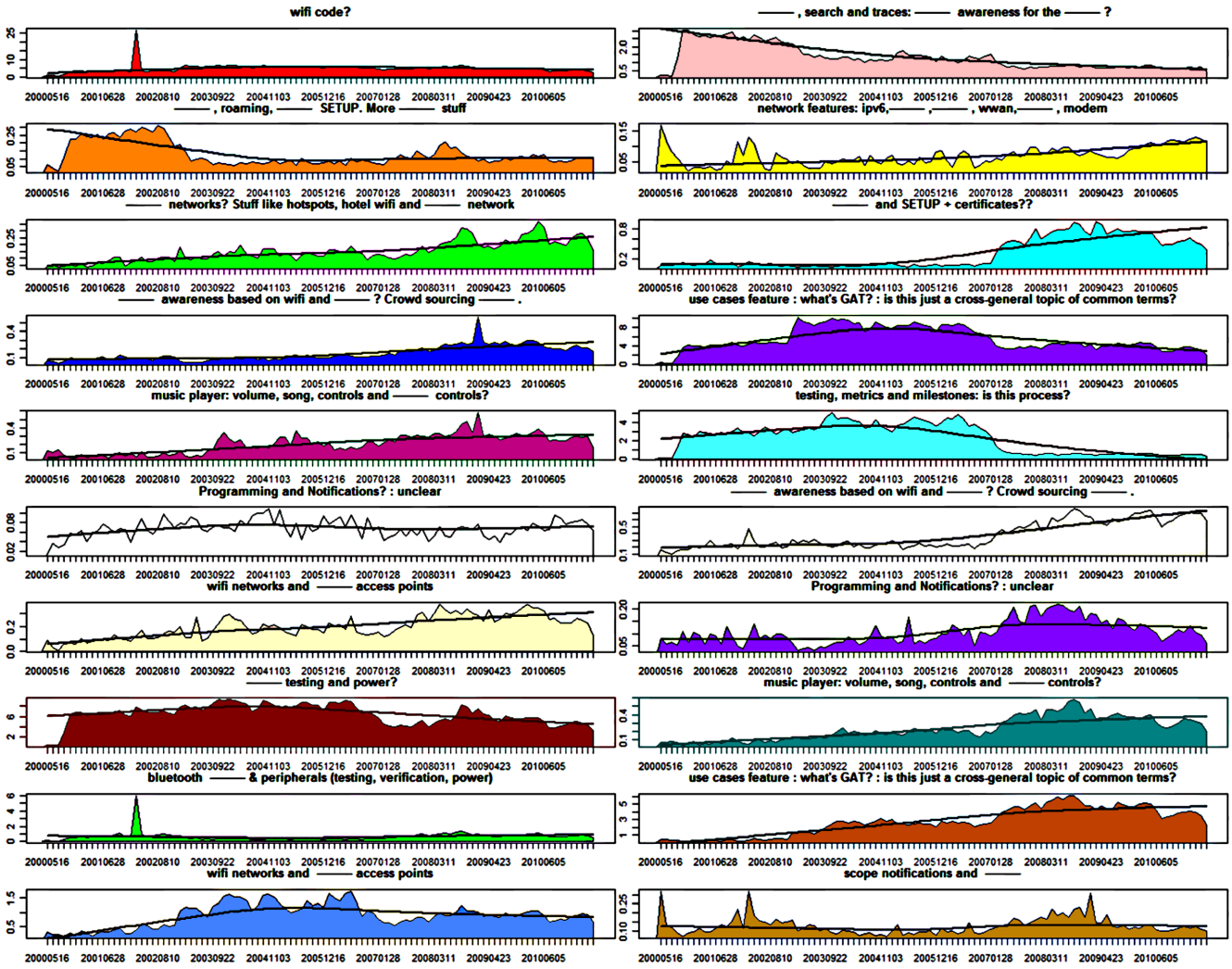


Fig. 2. Topics and Relevant revisions over time. X axis is time, Y axis is the average relevance per time window (greater is larger).

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.

B. Requirements Topic Mining

To apply LDA to requirements we preprocess the documents. We convert each specification to word distributions (counts of words per document) of stemmed words and remove stop words (common English stop words such as "at", "it", "to", and "the"). We provide these word distributions to our implementation of the LDA algorithm. LDA produces a requested number of topics from these processed requirements documents.

While authors such as Grant et al. [23] have proposed heuristic methods of suggesting the best number of topics, we took a more subjective approach. 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 topics that were extracted 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, and did not share excessive disjoint concepts or ideas. Our number of topics selection algorithm was like a manual 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. [10] reported similar results. We argue that if practitioners were following this methodology they would not want many

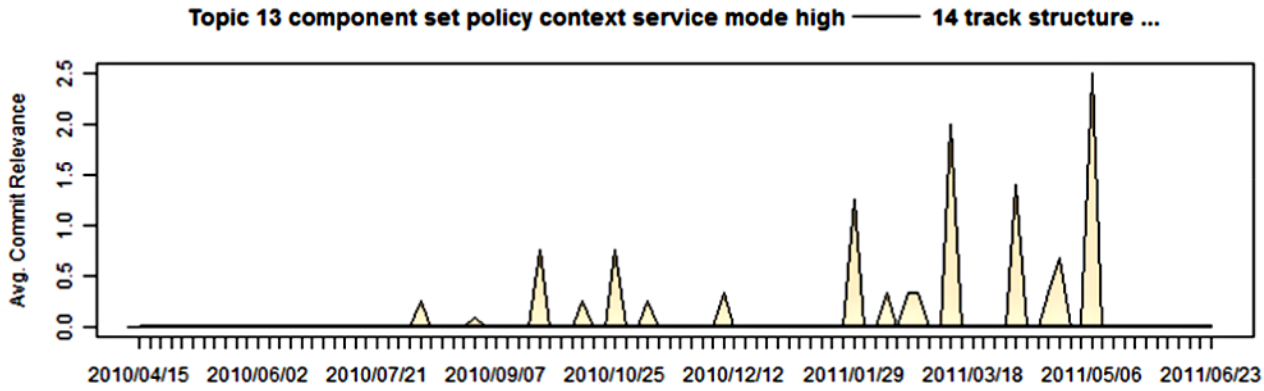


Fig. 3. Personal topic from 1 developer. Topics and Relevant revisions over time. X axis is time, Y axis is the average relevance per time window (greater is larger).

topics because it takes time to label each topic; our survey respondents took approximately 1 to 4 minutes (2 on average) for each topic. Thus we used LDA to produce 20 topics.

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. Labelling topics was difficult as there were many project specific terms that did not have a clear definition.

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).

C. 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 4000 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).

D. Relating Requirements Topics to Commits

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. 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 and used the intersection of the vocabulary (as we were inferring, not learning) shared by the requirements documents and topics.

Thus we intersected each commits words by the words in our topics and then inferred (via LDA inference) the relationship between the requirements topics and the changes over time. 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. We inferred the topics related to each change, leaving us with a document topic 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. The LDA inference technique was used to query bug reports previously by Lukins et al. [24]. We did not train on the commits because our goal was to use topics that were requirements relevant. Also, inference allows for fewer topic updates as requirements updates are less frequent than commits.

LDA inference produced a document-topic matrix 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. This allows can allow us to plot time-series of commits that are relevant to topics. In the next section we discuss how this matrix allows us to plot the relationship over time between requirements topics and commits.

E. 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, we had to present summaries of this effort to the users. We used a time-line overview, a topic-plot (see Figure 2), which showed the relevant commits over time. We also utilize our previous topics labels to label these plots as shown in Figures 2 and 3. 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 slice and dice 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 topic plots*, Figure 3 is an example of one of these 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.

Note that a single commit can be associated with multiple topics [23]. 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. Note that —— indicates a redaction, within this study we redacted some tokens and words that would disclose proprietary information. Within this figure we can see distinct behaviours such as certain topics increasing or decreasing in importance.

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

F. Qualitative Evaluation

Our goal was to determine the utility of the global revision topic plots and the personal topic plots (e.g. Figure 3). Thus we interviewed relevant stakeholders and developed a survey. Our interviews and surveys were designed using methodologies primarily from Ko et al. [25] and those described by the empirical software engineering research of Shull et al. [26] and Wohlin et al. [27], regarding interviews, surveys and limited study sizes.

1) PM and Developer Interviews: Our goal is to determine if these labelled topics are relevant to developers and program managers (PMs), all of whom are development engineers. 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 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.

2) 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 V-B.

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 developers 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:

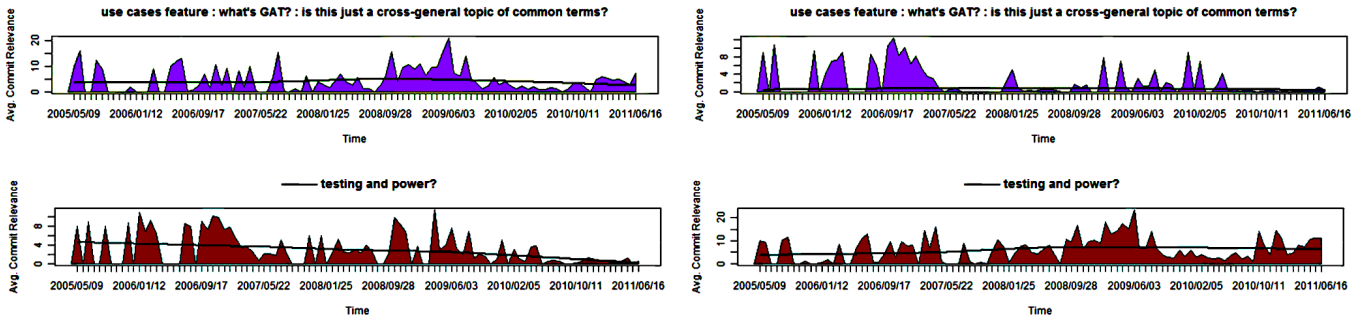
component set policy context service mode high 14 track structure check type hresult feature gat ap follow data dynamic.

G. Summarizing the results

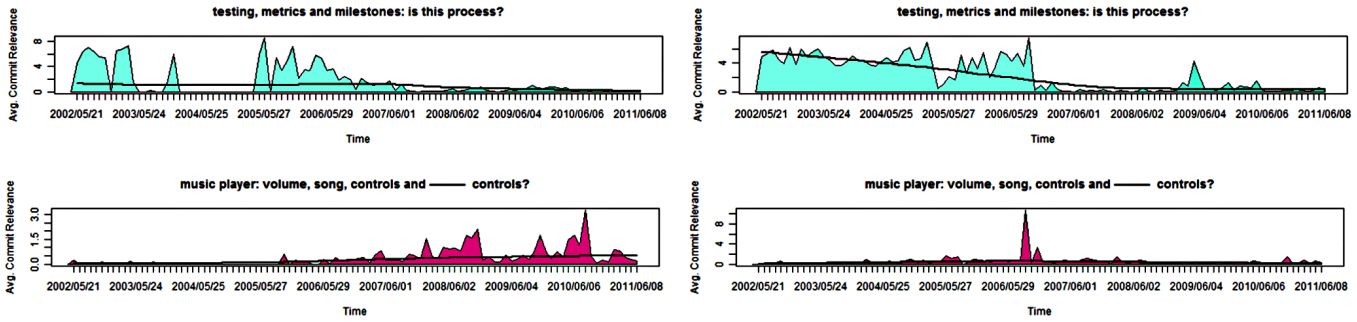
After the surveys were administered we discussed the results with each other, analyzed the data and collated our observations about the topic labelling, topic interpretation and if requirements topic-plots matched the developers' perception of effort.

IV. Topics of Requirements

Within this section we investigate the requirements topics and the commits and efforts that are relevant to these topics. We argue that these topics provide some traceability between requirements documents and the commits themselves, as well as provide a high-level overview of requirements relevant effort.



(a) Topic plots of 2 developers



(b) Topic plots of 2 teams

Fig. 4. Topic plots of 2 topics for 2 different developers and teams.

A. *Topic Plots of Requirements Relevant Effort*

Figure 2 depicts the topic commits relevant to 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 topic focus of the entire system over its lifetime. We can see the transition of focus on different requirements and topics as the system evolves.

Figure 2 gives us an overview of the relevant effort exhibited in a very large data-set. For instance, the 6th topic (3 down, right 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. The spikes in the plots are of interest (plots 7 and 9, 3rd and 4th down on the left side), as in Section V-A1 we found that a program manager indicated that the spikes were related to design change requests. Another example of interesting behaviour includes the use cases and testing topics (8th and 10th topics, 4 and 5 down on the right) became less important, information that would be useful to a manager.

While Figure 2 is a global plot that includes all commits in the project and is useful for high level managers, developers are more likely to be interested in more fine grained information about themselves and their teams and teammates, Figure 4a depicts the personal topic-plots of 2 developers (drawn only from commits made by those developers). By examining Figure 4a, we observe that the developers behaviour does indeed change over time, and each developer exhibits a different focus. For instance the behaviour in the plots on the topic, use case features, is distinctly different. Plots like Figure 4a illustrate that different authors focus on different topics in general. We found this was indeed the case by clustering similar developers, but these plots allow us to see where behaviour correlates and where it does not.

This is more apparent in Figure 4b, which depicts different teams rather than different authors. The we can slice these plots by authors and by organization to see what the particular focus an entity (team or author) had in terms of requirements topics. For *testing, metrics and milestones* in Figure 4b, the trend was similar but the behaviour was not. This manager level view provides a summary of a teams 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 results 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 observe global trends, personal trends or team-wise trends within these plots.

V. Qualitative Evaluation

In this section we validated with developers and managers if they 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.

A. Initial 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.

1) An Interview with a Program Manager: We interviewed a program manager 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). 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.

When shown a topic plot 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 our topic-plots mapped to the perception of the program manager.

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.

2) An Interview with a Developer who Writes Requirements and Design Documents: The 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 projects 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 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 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.

B. Survey Response

We administered surveys over email and in person. In person surveys were favored due to low response from the email survey. 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 a topic 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 3 and 4a, 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 given a summary of his development effort he could correlate his perception of what occurred with the topic plot provided: this topic plot matched his perception.

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 participant about a topic that was related to client server interactions.

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

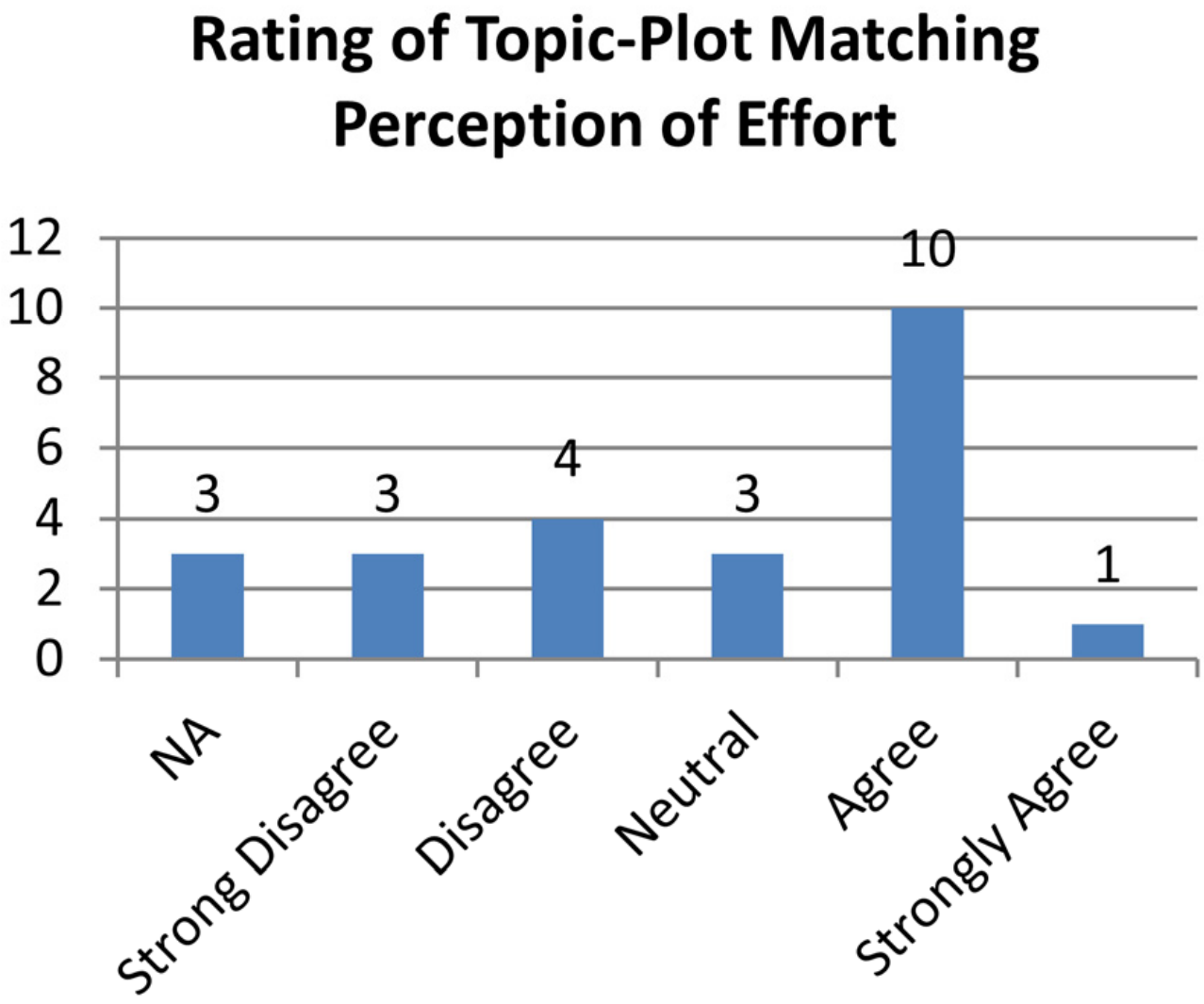


Fig. 5. Distribution of topic-plot perceptual ratings performed on two separate modules.

"This plot matches my perception of the effort that went into that topic and/or its features," 46% of topic-plot ratings were in agreement with this statement (see Figure 5).

C. Did Respondents Agree on Topic Labels?

One topic that we had labelled "wifi networks and software access points", had agreement between 2 of the respondents. One said: "Configuration of [software access point]", the other said " but really [it is] [software access point]. [In particular], the [user interface] related to configuring network broadcast, connectivity and sharing."

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.

VI. Discussion

A. 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.

1) Developers: Developers seemed to express difficulty labelling topics, but to be clear, many developers did not write the requirements or the design documents 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.

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.

3) Non-Experts: We consider ourselves to be experts in software, but not experts about the actual product we studied. Thus we relied on the experts to help us label the topics. At the same time we also labelled the topics. We examined all the topic labels and checked to see if there was agreement between our labellings and theirs. Of 46 separate labellings, 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.

This indicates that while non-experts can label these topics, there is inherent value in getting domain experts to label the topics.

B. 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 developers activity. If a plot did not match a developers perception, it seemed to be due to a lack of familiarity with the topic or the topic being unclear rather than the plot missing efforts that they expected: matching noise to noise produces noise. Many respondents indicated that we should talk to the team that produced the requirements we used. 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.

VII. 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 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 20 topic words. Thus multiple topics multiply the cost of interpretation.

VIII. Threats to Validity

Relevant *construct validity* threats include the fact we used only one large project 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. 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. Training and verbal administration of surveys can also bias results.

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 [25] qualitatively, as we observed repeated answers.

External validity is threatened by the fact 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.

IX. 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.

X. Conclusions

In conclusion, we conducted an evaluation of the commonly used practice of LDA topic analysis for traceability research with Microsoft developers, rather than students, in a large project with comprehensive requirements documents.

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. Many topics extracted from requirements were relevant to features and development effort. Stakeholders who were familiar with the requirements documents tended to be comfortable labelling the topics and identifying behaviour, but those who were not, showed some resistance to the task of topic labelling. Non-expert topic labelling suffered from inaccuracy.

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. 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 and 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 this work as a visiting researcher at Microsoft Research.

References

[1] A. Kozlenkov and A. Zisman, "Are their design specifications consistent with our requirements?" in *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, ser. RE '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 145–156.

[2] K. E. Wiegers, *Software Requirements*, 2nd ed. Redmond, WA, USA: Microsoft Press, 2003.

[3] N. Tillmann, F. Chen, and W. Schulte, "Discovering likely method specifications," in *Formal Methods and Software Engineering*, ser. Lecture Notes in Computer Science, Z. Liu and J. He, Eds. Springer Berlin / Heidelberg, 2006, vol. 4260, pp. 717–736.

[4] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina, "Goal-centric traceability for managing non-functional requirements," in *Proceedings of the 27th international conference on Software engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 362–371.

[5] M. Sabetzadeh and S. Easterbrook, "Traceability in viewpoint merging: a model management perspective," in *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, ser. TEFSE '05. New York, NY, USA: ACM, 2005, pp. 44–49.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[7] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 95–104.

[8] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya, "A theory of aspects as latent topics," in *Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, ser. OOPSLA '08. New York, NY, USA: ACM, 2008, pp. 543–562.

[9] T. Savage, B. Dit, M. Gethers, and D. Poshyvanyk, "Topicxp: Exploring topics in source code using latent dirichlet allocation," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ser. ICSM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–6.

[10] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Modeling the evolution of topics in source code histories," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11. New York, NY, USA: ACM, 2011, pp. 173–182.

[11] ——, "Validating the use of topic models for software evolution," in *Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, ser. SCAM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 55–64.

[12] A. Hindle, N. A. Ernst, M. W. Godfrey, and J. Mylopoulos, "Automated topic naming to support cross-project analysis of software maintenance activities," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11. New York, NY, USA: ACM, 2011, pp. 163–172.

[13] B. H. C. Cheng and J. M. Atlee, "Research directions in requirements engineering," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 285–303.

[14] B. Ramesh, "Factors influencing requirements traceability practice," *Commun. ACM*, vol. 41, no. 12, pp. 37–44, Dec. 1998.

[15] N. Ernst and J. Mylopoulos, "On the perception of software quality requirements during the project lifecycle," in *Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science, R. Wieringa and A. Persson, Eds. Springer Berlin / Heidelberg, 2010, vol. 6182, pp. 143–157.

[16] G. C. Murphy, D. Notkin, and K. J. Sullivan, "Software reflexion models: Bridging the gap between design and implementation," *IEEE Trans. Softw. Eng.*, vol. 27, no. 4, pp. 364–380, Apr. 2001.

[17] H. M. Sneed, "Testing against natural language requirements," in *Proceedings of the Seventh International Conference on Quality Software*, ser. QSIC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 380–387.

[18] S. P. Reiss, "Incremental maintenance of software artifacts," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 682–697, Sep. 2006.

[19] D. Poshyvanyk, "Using information retrieval to support software maintenance tasks," Ph.D. dissertation, Wayne State University, Detroit, MI, USA, 2008.
[20] C. McMillan, D. Poshyvanyk, and M. Revelle, "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*, ser. TEFSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 41–48.
[21] S. Konrad and B. Cheng, "Automated analysis of natural language properties for uml models," in *Satellite Events at the MoDELS 2005 Conference*, ser. Lecture Notes in Computer Science, J.-M. Bruel, Ed. Springer Berlin / Heidelberg, 2006, vol. 3844, pp. 48–57.
[22] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *Proceedings of the 11th Working Conference on Reverse Engineering*, ser. WCRE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 214–223.
[23] S. Grant and J. R. Cordy, "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*, ser. SCAM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 65–74.
[24] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Source code retrieval for bug localization using latent dirichlet allocation," in *Proceedings of the 2008 15th Working Conference on Reverse Engineering*, ser. WCRE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 155–164.
[25] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proceedings of the 29th international conference on Software Engineering*, ser. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 344–353.
[26] F. Shull, J. Singer, and D. I. K. Sjberg, *Guide to Advanced Empirical Software Engineering*, 1st ed. Springer Publishing Company, Incorporated, 2010.
[27] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.