Information and Software Technology 000 (2016) 1–14

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis

Jie Zou[b], Ling Xu[a,b,*], Mengning Yang[b], Xiaohong Zhang[b], Dan Yang[b]

[a] Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, Chongqing 400044, PR China
[b] The School of Software Engineering, Chongqing University, Huxi Town, Shapingba, Chongqing (401331), PR China.

ARTICLE INFO

Article history:
Received 27 March 2016
Revised 26 November 2016
Accepted 10 December 2016
Available online xxx

Keywords:
Non-functional requirements (NFRs)
Topic model
Latent Dirichlet allocation (LDA)
Stack Overflow

ABSTRACT

Context: As a vital role for the quality of software, non-functional requirements (NFRs) are attracting greater attention from developers. The programming question and answer (Q&A) websites like Stack Overflow gathered the knowledge and expertise of developers worldwide which reflects some insight into the development activities (e.g., NFRs), but the NFRs in the Q&A site are rarely investigated.
Objective: Our research aims to aid comprehension on the actual thoughts and needs of the developers by analyzing the NFRs on Stack Overflow.
Method: We extracted the textual content of Stack Overflow discussions, and then we applied the topic modeling technique called latent Dirichlet allocation (LDA) helping us to discover the main topics of the corpus. Next, we labelled the topics with NFRs by the wordlists to analyze the hot, unresolved, difficult NFRs, and the evolutionary trends which involves the trends of the NFRs focus and NFRs difficulty.
Results: Our findings show that (1) The developers mostly discuss usability and reliability while discussing less on maintainability and efficiency. (2) The most unresolved problems also occurred in usability and reliability. (3) The visualization of the NFR evolutions over time shows the functionality and reliability attract more and more attention from developers and usability remains hot. (4) The NFRs investigation in specific technologies indicates the quality is a similar concern among different technologies and some NFRs are of more interest as time progresses. (5) The research on NFRs difficulty in specific technologies shows the maintainability is the most difficult NFR. In addition, the trends of the NFRs difficulty over time in the seven categories signal that we should focus more on usability to address them.
Conclusion: We present an empirical study on 21.7 million posts and 32.5 million comments of Stack Overflow, and our research provides some guide to understand the NFRs through developers' eyes.

1. Introduction

Today, people are more concerned about software quality requirements with the increase in the scale and size of software. Non-functional requirements (NFRs) describe important constraints upon the development and behavior of software, they should be considered as early as possible, otherwise they may cause some latent problems on final product like unstable and low quality. It may become very complex and expensive to address them at later stages of software development, especially for large systems.

Programming question and answer (Q&A) websites, such as Stack Overflow, allows developers to ask questions to other developers and experts. It provides a new means for programmers to participate in social learning [1]. As one of the largest and most popular online Q&A forums, Stack Overflow gathers knowledge and expertise of developers on a wide range of topics. Also, many of them are related to NFRs, which are ignored by most of the previous research. The Stack Overflow discussions represent the actual thoughts and needs of developers from all over the world.

The data extracted from Stack Overflow contains millions of posts created by tens of thousands of developers. There are in total about 21.7 million posts and 32.5 million comments used in this paper. Analyzing the textual content of such a large knowledge repository poses a number of challenges (e.g., prohibits manual analysis and most conventional data mining techniques are far from being effective). In view of this, the topic model, Latent Dirichlet Allocation (LDA) [2], can help to understand the unstructured nature of the data which is written in natural language. We used topic modeling as a way to summarize them. Topic modeling

* Corresponding author.
E-mail address: xuling@cqu.edu.cn (L. Xu).

http://dx.doi.org/10.1016/j.infsof.2016.12.003
0950-5849/

is one technique that has been successfully applied in the past to summarize large corpora in many different fields including software engineering [3–6]. Previous works on topic analysis also show that extracting the topics can be useful to understand software maintenance [7, 8]. The topics extracted from the corpus summarized the key concepts in the corpus, such as source code, text descriptions, and commit comments. The topic model LDA can find the topics of the corpus. It creates multinomial distributions of the words to represent the topics.

Since the extracted topics are abstract and hard to understand, we need to devise appropriate labels for these topics to make it easy to use in discussion and analysis. The identified topics made sense to practitioners and matched their perception of what occurred [9]. We annotate them with the NFRs because the topic trends often corresponded to NFRs [10]. Through that we discover the focus and needs of the developers, present visualizations of NFRs that developers discussed during the development activities. Managers can also grasp the topic-related activities of the developers more easily.

In this paper, we extract the topics of the corpus using the topic model LDA, and then annotate the topics with the NFRs labels using a special wordlist. After that we analyze the NFRs in all discussions and specific technologies. The NFRs analysis in all discussions includes hot NFRs, unresolved problems, and evolutionary trends. The NFRs analysis in specific technologies includes hot NFRs, difficult NFRs, and the NFR difficulty trends. We divide the corpus into time-windows when extracting the topics of the corpus. We partition the corpus into 30 day periods (one month), and then applied LDA to each of these windows. We divide the corpus into time-windows rather than regard it as a whole because many topics are quite local. This also allows us to explore the evolution of the NFRs and the trends of NFRs difficulty over time. When annotating the NFRs, we utilize the standards of ISO9126 [11], which describes six high-level NFRs: maintainability, functionality, portability, efficiency, usability, and reliability.

We use the data posts from Stack Overflow, and in our analysis, we also distinguished between posts with or without comments to compare the results between the two settings involving single information sources or a combination of information sources. Our study aims to answer the following five research questions:

RQ1) What are the hot NFRs in all discussions and specific technologies?
RQ2) Which NFRs questions remain unanswered the most?
RQ3) What are the trends of the NFRs with respect to time?
RQ4) What is the most difficult NFRs the developers face in different specific technologies?
RQ5) How do the NFRs difficulty in specific technologies change over time?

By answering these questions, we want to provide many immediately useful details about the software development activities. Since Stack Overflow is worldwide and very popular with tens of thousands of developers, the discussions tendency on Stack Overflow are symbiotic with the world tendency. The analysis of hot NFRs and unresolved problems indicate the focus and needs of developers, give guidance to developers as to which NFRs they should focus on, and signals development supporters to which fields in software development activities needs their attention. The evolution of the NFRs provides the changes in developer focus as they change over time. It helps the software practitioners to understand the NFRs history and predict the NFRs tendency in their software development. It allows managers and maintainers to track the key aspects of development activities, commercial vendors to assess the adoption rate of their products, and to understand usage trends. The exploration in the specific technologies gives us insight to developer interests at a finer grain. This lets us have a more focused view about particular subfield. It also allows us to make a comparison between the different specific technologies (e.g. different programming languages: java, python, and android). Based on this we can find the relationships and differences between different categories. The hot-NFRs in specific technologies give us a finer indication for the developers work on specific fields. Moreover, the research provides software engineering researchers with immediate knowledge of some of the possible troublesome areas. The analysis of difficult NFRs and their trends helps the managers and researchers to determine where to increase resources and when to offer a bounty. It provides a way to analyze the most confusing aspects of particular technologies.

The remainder of this paper is organized as follows. Section 2 discusses the related work. Section 3 presents our approach including how we process the dataset and annotate the NFRs labels to analyze the activities. Section 4 shows the experimental results analysis. Section 5 provides the threats to validity. Section 6 presents a conclusion of our work.

This work extends our previous work [12]. The major extensions in this paper are the four refined explorations to investigate the NFRs in specific technologies, which give us insight to developer interests at a finer grain:

- We conduct an analysis of the hot-NFRs in specific technologies and the relations and differences among them.
- We explore the correlation between the NFRs importance and time in specific technologies.
- We investigate the approach to determine the difficulty of NFRs, and analyzed the difficult domains of NFRs the developers face in different specific technologies.
- We analyze the trends of NFRs difficulty over time in specific technologies.

2. Related work

In this section, we summarize the related work in three categories: the study of Q&A website, Stack Overflow; the study of NFRs; and the use of topic model LDA to study focus and trends in software engineering data.

2.1. Stack Overflow

A number of previous works have studied Stack Overflow. Treude et al. [13] used Stack Overflow to categorize its questions. Mamykina et al. [14] used Stack Overflow to identify its design features. Asaduzzaman et al. [15] researched the unanswered questions on Stack Overflow. They found that 7.5% of questions on Stack Overflow are unanswered and they want to help understand the delay in answering questions by providing a taxonomy. Bosu et al. [16] made an empirical investigation on Stack Overflow to understand its dynamics of building reputation. Bajaj et al. [17] used Stack Overflow data to study the misconceptions and common challenges among web developers. Jin et al. [18] studied quick trigger on Stack Overflow. Goderie et al. [19] proposed a tag-based approach to predict the response time in Stack Overflow. Calefato et al. [20] investigated how Stack Overflow users can increase the chance of getting their answer accepted. Their main findings were being fast and being nice when answering questions. Novielli et al. [21] applied a systematic sentiment analysis to the content available in Stack Overflow. Their investigation highlighted challenges of sentiment detection, and they found that affect is a quite complex phenomenon whose polarity is only one dimension of analysis. Anderson et al. [22] discovered the long-term value from community activity on Stack Overflow. They presented a valuable investigation on the foundations for reasoning about community processes in question-answering by considering the dynamics of

Please cite this article as: J. Zou et al., Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis, Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.12.003

the community activity. Their new understanding of the properties is also applicable to predicting the long-term value of a question and whether a question requires a better answer. Slag et al. [23] investigated One-day flies on Stack Overflow, they wanted to explain the situation why the vast majority of Stack Overflow users only posts once. Marder [24] proposed an econometric approach to study the badges and user behavior on Stack Overflow. Nasehi et al. [25] conducted a qualitative analysis of Stack Overflow posts to investigate what makes an effective code example. Honsel et al. [26] performed an evaluation of nine common myths about Stack Overflow posts. They wanted to explore whether the intuition of the developers is really true. Chowdhury and Hindle [27] proposed a classified approach to filter out off-topic Internet Relay Chat discussions. Their approach shows that similar data from different sources can be used for text classification. An empirical study to determine the needs and challenges developers face is provided by Li et al. [28]. In contrast to these efforts, we attempt to research from the perspective of NFRs specially, in order to provide some insight to understand the NFRs through developers' eyes.

Parnin et al. [29] made an empirical study of the crowd documentation for three popular APIs on Stack Overflow, including Android, GWT, and the Java programming language. Linares-Vásquez et al. [3] extracted hot-topics from mobile-development related questions of Stack Overflow. In contrast, we research on more and broader technology categories. We expect to discover more aspects of development activities.

2.2. NFRs

One of the first studies that investigated how to deal with NFRs in the software development process was presented by Chung and Nixon in [30]. They concluded that an NFR Framework [31] would be helpful for developers and users. With the review on the treatment of NFRs, Chung et al. performed a systematic and prominent investigation on NFRs in Software Engineering [32]. Mairiza et al. [33] studied the NFRs on three essential dimensions, including definition and terminology; types; and NFRs in various types of systems and application domains. Doerr et al. [34] presented an industrial study on NFRs using a hierarchical quality model. Glinz [35] surveyed the definitions of NFRs and discussed their problems, and then proposed some concepts for overcoming these problems. Umar and Khan [36] analyzed various techniques, practices and frameworks of NFRs for software development. Galster and Bucherer [37] categorized and specified NFRs for service-oriented systems. Ameller and her colleagues [38] made an exploratory study on how software architects consider the NFRs in practice. They focused on the NFRs terminology, ranking of types and tool support based on 13 interviews with software architects. With respect to model-driven development (MDD), Ameller et al. [39] integrated NFRs into the MDD process. Kugele et al. [40] introduced an integrated model-driven development process for deployment of real-time systems. Ahmad et al. [41] proposed an integrated approach for the modeling and verification of functional and non-functional requirements. Borg et al. [42] performed an investigation on real-world treatment of NFRs. They found that software development organizations mainly focus on functional requirements while NFRs are difficult to elicit. Damm et al. [43] proposed a so-called rich component model that integrates functional and NFRs based on UML. Cleland-Huang et al. [44] introduced a novel approach to detect and classify the NFRs by requirements documents. Eckhardt and his colleagues [45] presented discussions on the integration of NFRs in seamless modeling. With respect to the distinction between the NFRs and functional requirements, Eckhardt et al. [46] found that most "non-functional" requirements are really not non-functional. They state that many so-called NFRs can be handled similar to functional requirements. In contrast to these efforts, our paper presents an empirical study on NFRs by mining the textual content of the global Q&A site Stack Overflow using topic analysis. We analyze the NFRs focus, NFRs difficulty, and their trends over time, which are ignored by most of other research.

Table 1
Categories of questions.

Category	Count	Related Stack Overflow Tags
Java	716,908	java, java-ee
Python	339,651	Python
Android	564,955	android
CSS	262,445	css, css3
SQL	532,121	sql, mysql, sql-server, postgresql, databases
Version Control	69,397	git, mercurial, svn
Build Tools	35,927	ant, maven

2.3. Latent Dirichlet Allocation

LDA is one technique that has been previously successfully employed in Software Engineering [47-49]. Barua et al. [3] discovered the topics from the textual content of Stack Overflow by a statistical topic model. Through a set of words within the topics, they manually associated the topics to definite technologies, such as mobile application, web development, and data management. Then Linares-Vásquez et al. [4] and Rosen et al. [5] focused on mobile development-related discussions from Stack Overflow using LDA. Allamanis and Sutton [50] presented a topic modeling analysis to associate the questions with programming concepts and identifiers. Our work is similar to theirs because of the LDA model, but is different from the type of information analyzed. We keep our eyes on the NFRs the developers are concerned based on the Stack Overflow discussions.

3. Data and approach

In this section we describe how to annotate the NFRs. Our approach consists of three key steps. Firstly, we extract the data from Stack Overflow, and then preprocess the extracted data. Secondly, we construct a topic model LDA to summarize the topics of the corpus. Finally, we label the topics with the NFRs by our wordlists. The overall process of our approach is shown in Fig. 1.

3.1. Data extraction process

To address the above research questions, we used the posts and comments of the Q&A site Stack Overflow from July 31, 2008 to September 14, 2014 provided by the MSR challenge [51], Stack Overflow is a global site which features questions and answers on a wide range of topics in computer programming. Since the original data is organized in the form of an XML file containing a lot of redundant information, we used the java SAX parser to extract the "title" and "body" of the posts and the "text" of the comments, totaling about 21.7 million posts and 32.5 million comments. The example (#141) of Stack overflow posts storage in the "Posts.xml" is shown in Fig. 2. For our analysis, to the first research question, we use two types of corpus. The first is the "title" and "body" of the posts, and comments when they exist. In which the posts include the Stack Overflow questions and answers. We also compare the results between the "title" and "body" of the posts with or without the "text" of the comments. The second is the categories data extracted from posts according to Table 1 from [50]. For the RQ2, we extract the "title" and "body" of the unanswered questions from the posts, totaling about 921 K. We treat the question as unanswered question if the count of answers for the question is 0.

Please cite this article as: J. Zou et al., Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis, Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.12.003

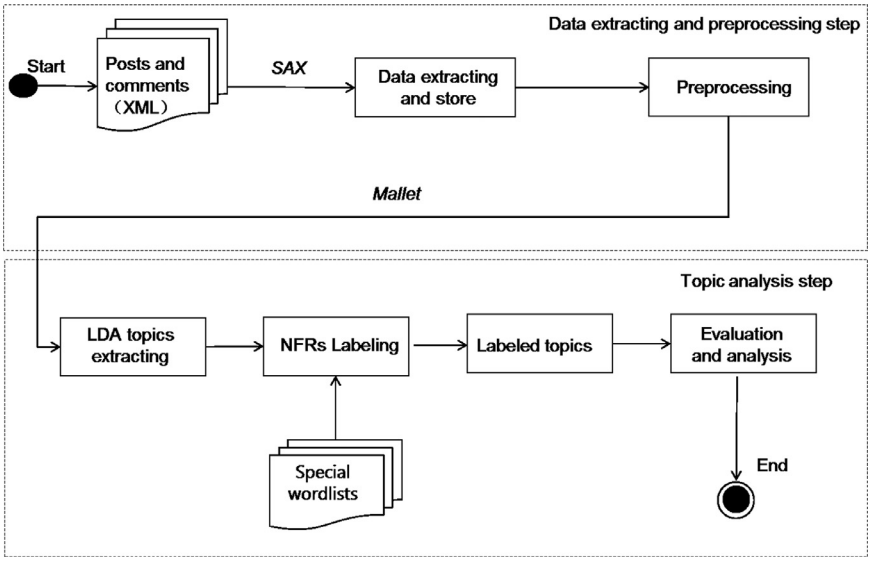


Fig. 1. An overview of our research methodology.

<row Id="141" PostTypeId="2" ParentId="129" CreationDate="2008-08-01T16:47:54.927" Score="1" Body="<p>Another tool to try would be the SQLMaestro suite - http://www.sqlmaestro.com It is a little tricky nailing down the precise tool, but they have a variety of tools, both free and for purchase that handle a wide variety of tasks for multiple database platforms. I'd suggest trying the Data Wizard tool first for MySQL, since I believe that will have the proper "import" tool you need.</p>" OwnerUserId="71" LastActivityDate="2008-08-01T16:47:54.927" CommentCount="0" />

Fig. 2. An example of posts.

In the RQ3, we use the "title" and "body" of the posts and the "title" and "body" of the unanswered questions to explore the NFRs trends in all discussions, and use the categories data to explore the NFRs trends in specific technologies. For the RQ4 and RQ5, the dataset we used is also the categories data extracted from posts.

Fig. 3 shows the detailed data for each month (period), with the month as the x-axis, for example, Sep-08 means September 2008, and the number of posts or comments as the y-axis, the highest reaching 0.95 million. Fig. 3(a) shows the overall dataset, and Fig. 3(b) shows the data of each specific technology with each month.

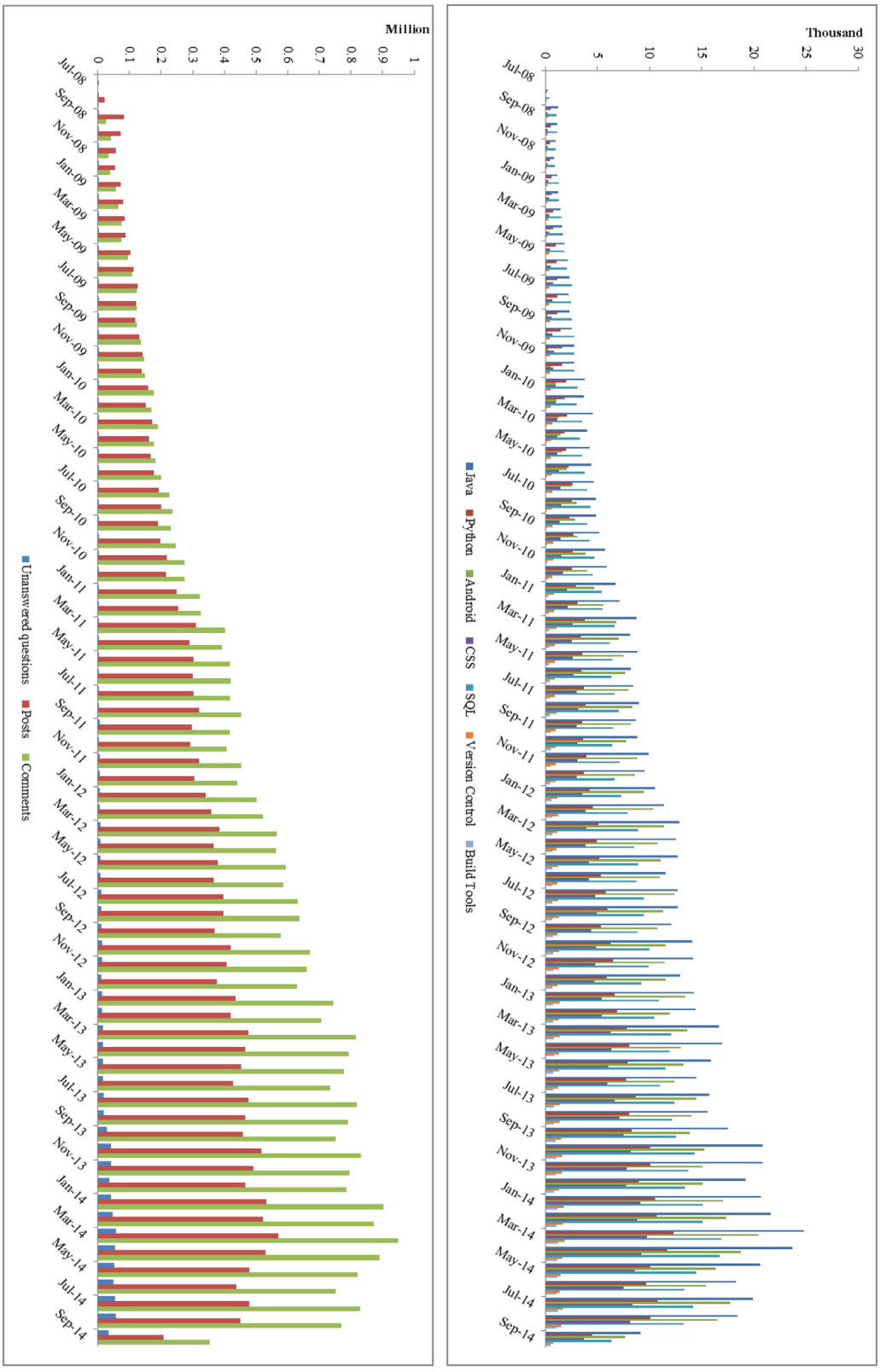
After data extraction, we preprocess the data as follows. At first, we remove the periods whose number of posts is less than 100, since too few posts are useless for the analysis. For example, in the month of July 2008, there are only 7 posts. This is because of the fact that only one day (July 31) of discussion is included in the dataset. Next, a few of remaining preprocessing steps are carried out to further refine the information. They comprise: tokenization, stop words removal, and case unification.

3.2. Topic modelling

To help understand the Stack Overflow discussions, we apply the topic model LDA [2] to summarize the topics of the corpus. LDA has been recognized as one of the best techniques for finding the topics of discussions in natural language text documents. It has been applied to various software engineering research questions, such as component assignment [47], software defect prediction [52], bug localization [48], and software change message classification [53].

In LDA, the topic is the conditional probability distribution of words in the vocabulary. It uses word frequencies and co-occurrences of frequencies in the document to build a model of related words [54]. That is, LDA creates topics when it finds there are sets of words that tend to co-occur frequently in the documents of the corpus. The words in a topic are usually semantically related. For example, a topic that contains the words "http" "href" "org" "html" (because these words occur together frequently in documents of the corpus), indicates that this topic is related to URLs. Given a set of documents, LDA infers the topics and topic memberships for each document by using machine learning algorithms.

In this paper, we construct the topic model LDA by MALLET [55], which is an implementation of the Gibbs sampling algorithm [56]. It requires various modeling parameters available as input, including the number of topics parameter K, the number of iteration N, the Dirichlet parameters α and β. The number of topics K is a user-specified parameter that controls the granularity of the discovered topics. The number of iteration N signifies the iterations of Gibbs sampling. α is a hyper-parameter that controls how much fuzziness is enabled in a topic's distribution across words, where the hyper-parameter β controls how much fuzziness is enabled in a topic's distribution across documents. In our experiment, we choose the number of topics as 20 for each period because duplicate words from different topics are infrequent when $K = 20$, it is an appropriate medium granularity value for our NFRs analysis [7] [10]. Also, we use the default settings $N = 1000$, $\alpha = 2.5$, $\beta = 0.01$ [57]. The output of LDA is a matrix M where rows correspond to the K topics of posts or comments and columns correspond to the words of the topics.



(a) The overall data (b) The category data

Fig. 3. The detailed dataset for each month.

Table 2
The NFRs and their associated wordlists.

Label	Related terms
Maintainability	maintainability, modular, decentralized, encapsulation, dependency, interdependent, understandability, modifiability, modularity, maintainable, maintain, stability, analyzability, changeability, testability
Functionality	functionality, accuracy, correctness, vulnerability, secure, accurate, vulnerability, vulnerable, trustworthy, malicious, policy, buffer, overflow, secured, certificate, exploit, compliant, functionality, practicality, functional, suitability, interoperability, accuracy, compliance, security
Portability	portability, transferability, interoperability, documentation, internationalization, i18n, localization, l10n, standardized, migration, specification, portability, movability, movableness, portable, installability, replaceability, adaptability, conformance
Efficiency	efficiency, optimization, fast, slow, faster, slower, penalty, factor, sluggish, optimize, profiled, performance, efficiency, efficient, “time behavior”, “resource behavior”
Usability	usability, flexibility, interface, screen, user, friendly, convention, human, default, click, guidelines, dialog, ugly, icons, ui, focus, feature, standard, convention, configure, menu, accessibility, gui, usability, serviceability, serviceableness, usableness, useableness, utility, usefulness, serviceable, usable, useable, learnability, understandability, operability
Reliability	reliability, failure, error, redundancy, fails, bug, crash, stable, stability, integrity, resilience, dependability, dependableness, reliability, reliableness, responsibility, responsibleness, dependable, reliable, maturity, recoverability, “fault tolerance”

Table 3
Some of the sample topics, NFRs labeling, and the examples of posts.

Topics (top 40 words)	NFRs labeling	Example of posts
error pre code path windows module node app js failed install modules installed stable errors directory linux version unreliability configure data message system log program exe os running warning npm expected usability fails bin common missing custom unable var cmd ...	Reliability, Usability	#2140744: Git Reliability and Usability in a centralised workflow ... <p>As per the Git client we used, we started using the e-git Eclipse plugin. But it didn't seem to be reliable and usable enough, ... Regardless of the client used (e-Git or TortoiseGit), this worries me about the robustness and reliability of the tool, ...
time memory bit don performance start times cache single solution lot real takes difference multiple simple space question answer random faster fast loop step algorithm program speed pretty objects calls stack operations actual hard slow won storage amount simply idea ...	Efficiency	#4018568: Help In Learning Algorithm Basics ... I don't really understand it clearly when it comes to checking the efficiency/performance of different algorithms using these notations. I understand Logarithms very well but the way they are been used in relation to checking algorithms performance makes me mad.... #3947384: Rails Users has_one:profile vs. show page<p>I have a best practice/efficiency/performance issue. ...

3.3. NFRs labeling

In order to analyze the NFRs, we need to annotate these topics with NFRs labels. To the taxonomy of the NFRs, we utilize the ISO quality model, ISO9126, which describes six high-level NFRs: maintainability, functionality, portability, efficiency, usability, and reliability. After extracting 20 topics per period by LDA, we associate each NFR with a list of keywords, called wordlists, to label each topic with the six high-level NFRs automatically. Table 2 shows the wordlists[1] we used, which are specific to the software field. These are taken from [10] where they are called exp2. The wordlists are project independent which are based on Boehm's quality model [58] and McCall's quality model [59]. We labelled the topics with an NFR where there was a match between a word in the corresponding wordlist and the same word somewhere in the frequency distribution of words that constitute the topic. Each topic can be label with more than one NFR if there is multi-NFRs match. Also, the topics are labeled with “none” if there is no match, which signifies this topic is not associated with any of the NFRs. Table 3 illustrates some of the sample topics, their NFRs labeling, and the examples of posts which contain these topics as dominated topics.

3.4. Metric

After labeling NFRs, we obtained the labeled topics to analyze the NFRs. To help us answer the research questions, we define the metric R_{NFRs} to indicate the rate of NFRs. The R_{NFRs} metric measures the proportion of topics labeled by the corresponding NFRs.

$$R_{NFRs} = \frac{1}{K * M} \sum_{i=1}^{M} N_i \quad (1)$$

Where K is the number of topics extracted by the LDA model in each period (month). M is the total number of months in the corpus. N_i is the number of topics labelled with the corresponding NFRs in the ith month. For example, if the number of topics in each month is 20, the total number of months is 5, and the numbers of topics labeled with an NFR (e.g., usability) from the first month to fifth month are 8, 9, 7, 5, and 11, then usability has a R_{NFRs} metric of 40%. It means 40% of all topics contain this NFR (usability). The R_{NFRs} metric allows us to measure the relative popularity of a certain NFR.

3.5. Creating a validation corpus

To evaluate the annotated results, we invited four PhD candidates who research in software engineering to label the topics manually as a validation set. The annotators label the data from Sep 2013 to Aug 2014, totaling 12 months. The labels they used were also one or more of the six NFRs of ISO9126. They could also label the topics with “none” if they though there are no NFRs associated with the topics. When labeling, they looked at the data, consisting of the frequency distribution of words that constitute the topic and the original posts (full text) related to the topic being annotated, using their expertise and then suggested the appropriate label to the period's topics. For example, a topic of Sep 2014 consisting of the word “code exit view error null method init custom button click ui fault stable input”, they tagged the topic with usability and reliability. The annotating includes two steps. Firstly, all the four annotators annotated the data from Sep 2013 to Nov 2013

[1] http://softwareprocess.es/y/neil-ernst-abram-hindle-whats-in-a-name-wordlists.tar.gz.

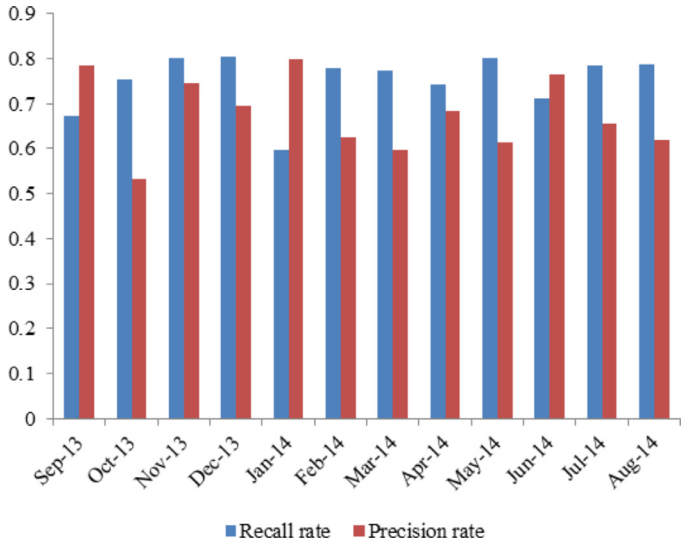


Fig. 4. The recall rate and the precision rate of our NFRs labeling.

in order to evaluate inter-rater reliability. Secondly, each of them labeled the data (from Sep 2013 to Aug 2014) for three months and did not annotate each other's annotations as a validation corpus.

4. Analysis of the results

4.1. Accuracy of evaluation

To answer the problem of how well our approach works, we use recall and precision rates as standard metrics to evaluate the accuracy of the NFRs labeling. We run our approach on the test set, which is the post data from Sep 2013 to Aug 2014. Next, we compute the recall rate and precision rate with the results of the manual validation set.

$$RecallRate = \frac{N_{detected}}{N_{total}} \quad (2)$$

$$PrecisionRate = \frac{N_{detected}}{N_{detectedall}} \quad (3)$$

Where $N_{detected}$ is the number of correct NFRs labels (the NFRs label matches the manual annotation), N_{total} is the total number of the manual NFRs labels in the testing set, $N_{detectedall}$ is the total number of NFRs labels in the experimental results using our approach (including the correct and incorrect). For example, if the topic is labeled with usability, efficiency and functionality by our approach, and the manual validation set labels the topic with usability, efficiency and reliability, $N_{detected}$ is 2 (usability and efficiency), N_{total} is 3 (usability, efficiency, and reliability), $N_{detectedall}$ is 3 (usability, efficiency, and functionality), so that the recall rate is 2/3≈66.7%, the precision rate is 2/3≈66.7%.

Fig. 4 shows the recall rate and the precision rate of our results. For each period (month), we calculate a recall rate and a precision rate. From Fig. 4, we see that the highest recall rate and precision rate of our results reaches 80%, averaging 75.9% and 68.7% respectively.

To determine inter-rater reliability of four annotators, we also calculate the Fleiss' kappa using three months data labeled by four annotators together. Fleiss' kappa is used for reliability measures, with values of 0.00 to 0.20 considered slight, 0.21 to 0.40 fair, 0.41 to 0.60 moderate, 0.61 to 0.80 substantial, and 0.81 to 1.00 almost perfect agreement [60]. Table 4 describes the Fleiss' kappa of our per-topic annotations for each NFR. We evaluated inter-rater reliability using each NFR because a topic could be labeled with more than one NFR.

Table 4
Inter-rater reliability for each NFR.

	Fleiss' kappa
Efficiency	0.6955
Functionality	0.6504
Maintainability	0.6175
Portability	0.5699
Reliability	0.6315
Usability	0.6376
None	0.7957

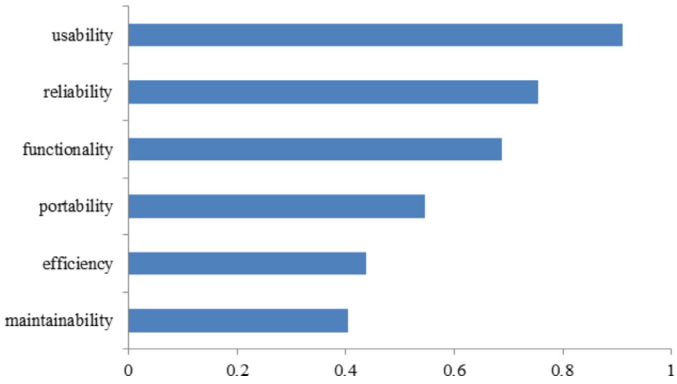


Fig. 5. The rate of different NFRs using posts.

As shown in Table 4, we see that six out of seven Kappas (efficiency, functionality, maintainability, reliability, usability, and none) are between 0.61 and 0.80, and the Fleiss' kappa of portability is between 0.41 and 0.60. That is, most of the NFRs' inter-rater reliabilities are substantial, only the inter-rater reliability for portability is moderate. It indicates that there is good agreement on the annotations of the four raters. Their annotations for the topics are relatively consistent. The reasons for the moderate inter-rater reliability for portability might be the expertise of four annotators about portability is various since the manual labeling is based on their personal experience. When labeling, they may tend to have different levels of understanding about the posts related to portability. Another explanation of the moderate result for portability may be that some unclear posts cause the ambiguity. We recommend that future annotators attend a consistent training and discuss how and when an annotation is appropriate.

4.2. Results RQ1

For research question 1, we investigate which NFRs are discussed most and least frequently to explore the hot NFRs the developers focus on. We research this from two aspects. Firstly, we discover the hot-NFRs in all discussions. Secondly, we investigate the hot-NFRs in specific technologies.

Fig. 5 shows the rate of different NFRs in all discussions which uses the data posts, where the x-axis is the R_{NFRs}, the y-axis is the corresponding NFRs. From Fig. 5, we see that the labels with the most topics are usability and reliability, and then functionality and portability. We did not see many results for efficiency or maintainability. That is, the NFRs the developers discussed most frequently are usability and reliability, and the least they discussed are efficiency and maintainability. When they are coding, they mainly focus on usability, and then reliability and functionality. Contrarily, they pay less attention to efficiency or maintainability. This indicates that most of developers worldwide tend to discuss usability of software products rather than maintainability. It also suggests the software vendors should attach greater importance to the usability of software products. In addition, we also compare the results using posts only with the dataset which contains the posts

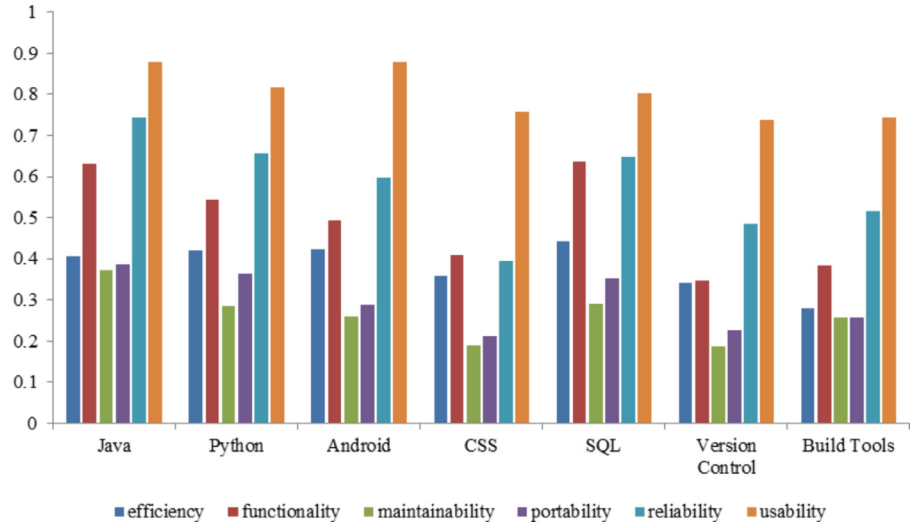


Fig. 6. The rate of different NFRs in seven specific technologies.

and comments. After analysis, we find that the relative distribution of the NFRs using posts and comments is almost the same as the results of using only the posts. The usability and reliability are also the highest, and then functionality and portability. The lowest are maintainability and efficiency. It means that the results are equal analyzing the topics with and without comments.

As most developers work on one or more specific technologies, they might be interested in the NFRs that focus on the specific technologies. Except to the hot-NFRs in all discussions, we also extracted the data of seven specific technologies as to explore the relation and differences of the NFRs focusing on different sub-domains. We would like to know whether some technologies discuss certain quality requirements more than others. This affects the project manager, especially a system manager, when confirming the initial design goals, or to track the progress on that quarter's objectives. Fig. 6 shows the rate of different NFRs in seven specific technologies. From this figure, we see that the label with the most topics is usability, and then reliability and functionality in the seven categories. The portability and maintainability do not appear very often in topics. All those developers focus most on the usability, and less on the maintainability, no matter which categories they are engaged. The developers value usability more than other NFRs when they are coding. The relative trends of six NFRs in seven categories are almost the same and the gap between the rates of each NFR on different technologies is also small. This suggests that software quality is of similar concern among different technologies. And we also see that the relative trends of six NFRs in Fig. 5 and Fig. 6 are almost the same. It suggests that the technology does not influence on NFR to be or not to be hot.

4.3. Results RQ2

For research question 2, we analyze the unanswered questions from Stack Overflow to explore the unresolved problem domains. Through this, we can provide more assistance to the developers. Fig. 7 shows the rate of different NFRs about unanswered questions. From this figure, we see the most topics remain unanswered are labeled with usability and reliability, and then functionality and efficiency. The fewest are portability and maintainability. This suggests that the problems remain unresolved are mostly in the usability and reliability domains. As to portability and maintainability, they usually can work this out by themselves, or we can say they rarely meet the problems of portability and maintainability. That is to say there are possible troublesome areas in usability and reliability, to offer more help to the developers, we should concern ourselves more with the issues of usability and reliability. Current processes and documentation should be improved by taking these areas into greater consideration and further research on them could also help improve the software development processes.

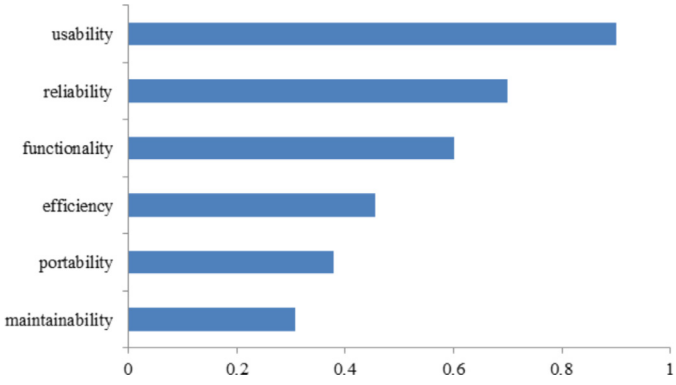


Fig. 7. The rate of different NFRs about unanswered question.

Table 5
Percentage of different NFRs labeling.

	Percentage
Multi-NFR	92.5%
One NFR	4.9%
None	2.6%

4.4. Results RQ3

Since the RQ1 concludes that the results of using posts only and using posts in combination with the comments are almost the same, in this research question, we use only posts data to cut down the data size. We label the topics of the posts using our approach to explore the trends of NFRs, including the trends of NFRs in all discussions and the trends in specific technologies. Table 5 lists the percentage of different NFRs labeling and from the table we see that most of the topics are labeled with more than one NFR, about 92.5%. The topics labeled with only one NFR is about 4.9%, and about 2.6% labeled with "none".

Please cite this article as: J. Zou et al., Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis, Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.12.003

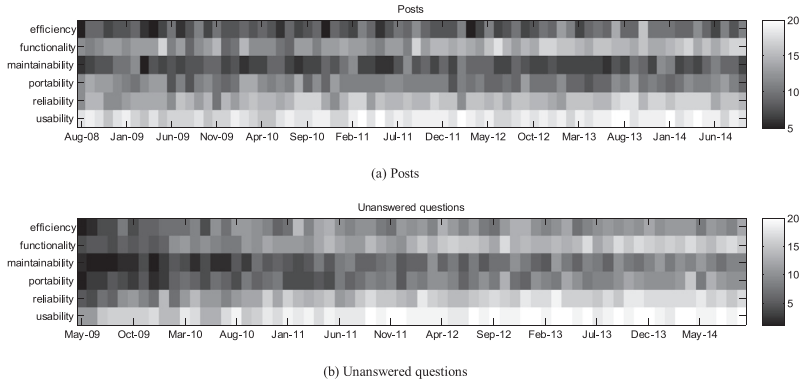


Fig. 8. The frequencies of different NFRs over time.

Fig. 8 shows the gray-scale image of the time-lines of the NFRs frequencies in all discussions. Each cell represents a 30-day period. Grid cell intensity (saturation) is mapped to label frequency which indicates the number of topics labeling the respective NFRs relative to the number of topics in that month. A deeper color indicates the smaller frequencies (white is the most, black is the least). Visualizing the results allows us to explore the data from different points of view to address different questions. From this figure, we can see the visualization of the evolution of each NFR with time, and we also can see the hot or not hot NFRs during a certain period. Fig. 8(a) shows the results of the posts, and we see that all six NFRs change over time, this suggests that developers shifted focus with respect to time when discussion. However, their trends are not the same. We observe that the efficiency, functionality and the reliability rise slowly while the maintainability is up and down as time goes on. At the same time the portability drops off. In addition, usability is almost stable at a high frequency. Fig. 8(b) shows the results of the unanswered questions. Four NFRs, functionality, maintainability, portability, and reliability increase with time since these NFRs have been more intensely shaded, the efficiency fluctuates and usability remains almost stable. From Fig. 8(a) and (b), we see that functionality and reliability are not only increasing for posts but also for unanswered questions. Moreover, in both cases usability was always at a high frequency throughout the lifetime. These suggest that functionality and reliability will increase the attention of the developers and the usability will remain hot in the next few years. There are also some peaks in Fig. 8, this shows that the maintenance activity is not necessarily strictly increasing or decreasing with time.

Also, we explore the correlation between the importance of software quality requirements and time in specific technologies. We want to explore if the NFRs are of more interest as years progress in specific technologies. For this question, we divide the datasets into 6 non-overlapping frames (or windows) by time, where each frame contains 12 months posts, to explore the NFRs trends from coarser time granularity in specific technologies. For example, Frame 0 contains the bug reports generated from Aug 2008 to Jul 2009; Frame 1 contains the bug reports generated from Aug 2009 to Jul 2010, Frame 5 contains the bug reports generated from Aug 2013 to Jul 2014. After that, the rate of labelled topics is obtained. Fig. 9 shows that the rate of different NFRs over time in seven specific technologies. From Fig. 9, we observe that the rates of NFRs are generally increasing with time for the seven categories. To further explore the relation between the NFRs and time, we calculate their Spearman's coefficients. We propose the null hypothesis that there might be a correlation between the importance of software quality requirements and the time. The results of Spearman's coefficients are shown in Table 6. When $n = 6$ (n is the number of observations), the critical values for Spearman's rank correlation coefficient is 0.829 with the significance level is 0.05 [61]. When $n = 5$, the critical value is 0.9. In Java, only the efficiency is below the critical value (0.829). In Python, the efficiency and usability are below the critical value (0.829). In Android, the usability is below the critical value (0.829). In CSS, Version Control and Build Tools, all the NFRs are above the critical value (0.829 for CSS and Version Control, 0.9 for Build Tools). In SQL, the functionality, maintainability and usability are above the critical value (0.829). In the seven categories, all the functionality and maintainability and most of the portability and reliability (six out of seven categories) are above the critical value. This indicates the developers are concerned these NFRs more and more with time. It suggests that the software quality requirements are of more interest as time progresses, which signals that the managers, developers and researchers should pay attention to the NFRs.

4.5. Results RQ4

For research question 4, we would like to know whether some NFRs are more difficult than others in the different specific technologies. Finding the most difficult NFRs will help software engineering researchers determine the most difficult area to work on. To determine the difficulty, we examine how likely it is for those issues to be successfully answered. As shown in Table 7, we apply four metrics to determine the difficulty of a topic from [5], including the "mean time", "median time", "% Accepted", and "Avg. # of Ans". For a question posted on Stack Overflow, the questioner would accept the answer as an accepted answer when it meets the questioner's satisfaction. We study the time it takes for these questions to receive an accepted answer by subtracting the creation dates of the accepted answer and the question posts. Next, we calculate the mean time and median time of questions in a topic. After obtaining all the metrics, we compute the difficulty of each NFRs by Ensemble Averaging and a linear combination using the NFRs labels of topics.

Fig. 10 shows the difficulty of different NFRs in seven specific technologies. From Fig. 10, we see that the maintainability and reliability are more difficult than others in Java, usability and portability in Python, maintainability and reliability in Android, maintainability and reliability in CSS, portability and maintainability in SQL, maintainability and functionality in Version Control, maintainability and efficiency in Build Tools. In Java and Android, the more difficult NFRs are similar. Two of them are maintainability and reliability. This signals the difficulty might be similar when the technologies are similar. In the seven categories, five out of seven of the most difficult NFR is maintainability. The average difficulty value of maintainability of the seven categories is also the highest. This signals that maintainability is the most difficult NFR among the six NFRs. These questions related to maintainability had

Please cite this article as: J. Zou et al., Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis, Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.12.003

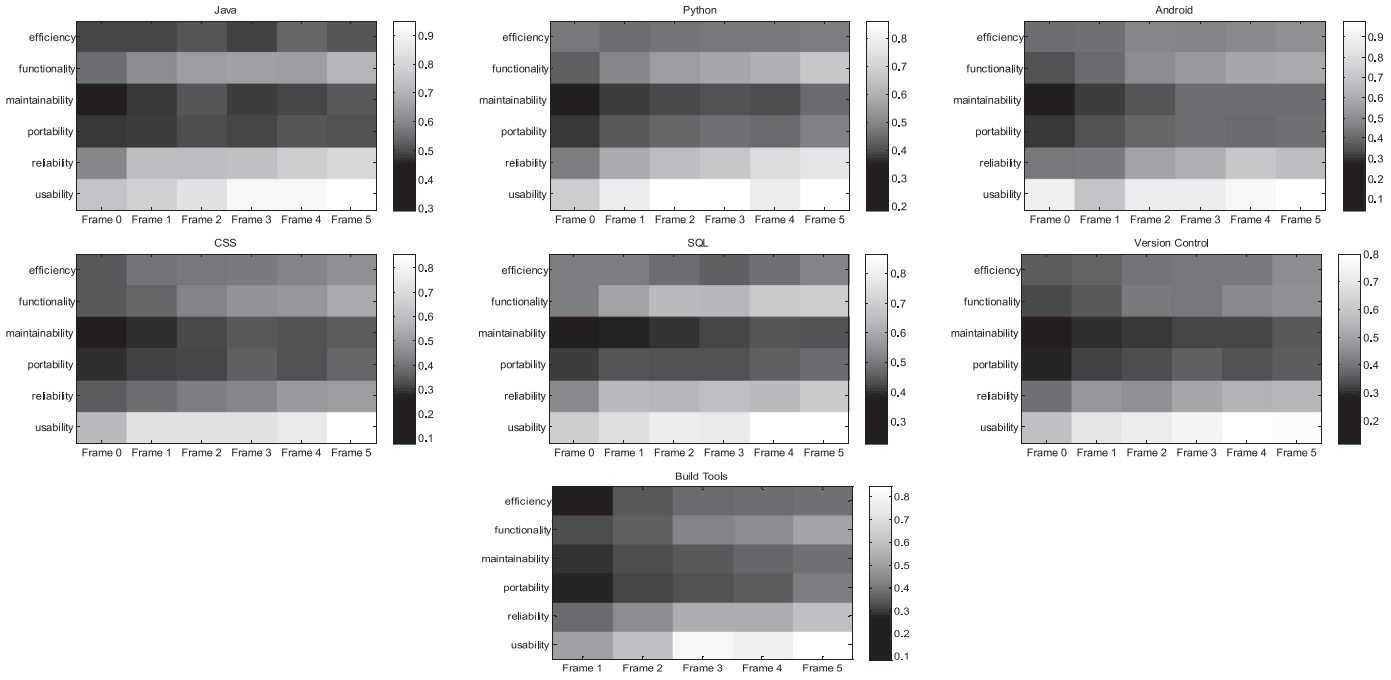


Fig. 9. The rate of different NFRs over time in seven specific technologies.

Table 6
The results of Spearman's coefficients.

Categories	NFRs	Spearman's coefficients	Categories	NFRs	Spearman's coefficients
Java	efficiency	0.464	Python	efficiency	0.638
	functionality	0.899		functionality	1
	maintainability	0.829		maintainability	0.943
	portability	0.886		portability	0.943
	reliability	0.943		reliability	1
	usability	0.986		usability	0.696
Android	efficiency	0.943	CSS	efficiency	1
	functionality	1		functionality	1
	maintainability	0.943		maintainability	0.943
	portability	0.943		portability	0.943
	reliability	0.943		reliability	1
	usability	0.638		usability	0.942
SQL	efficiency	0.086	Version Control	efficiency	1
	functionality	0.943		functionality	0.943
	maintainability	0.943		maintainability	0.986
	portability	0.812		portability	0.829
	reliability	0.812		reliability	0.943
	usability	0.886		usability	0.943
Build Tools	efficiency	1			
	functionality	1			
	maintainability	1			
	portability	1			
	reliability	0.9			
	usability	0.9			

Table 7
The metrics and their descriptions.

Metrics	Description
Mean Time	Mean time the questions takes to receive an accepted answer
Median Time	Median time the questions takes to receive an accepted answer
% Accepted	the percentage of questions posted on Stack Overflow that received accepted answers
Avg. # of Ans	the average number of answers the questions in a topic receive

an above average time to receive accepted answers, places in the bottom half for the average percentage of questions with accepted answers, and also below average number of answers to the question. We find that the easiest NFR in the following classifications is portability since its average number in seven categories is the lowest. This signals that the software practitioners should assign more human resources and material resources on maintainability and less on portability. We also find that the most difficult NFR in Python is usability. This indicates that the most popular NFR (usability, Results RQ1) also might tend to be the most difficult NFR.

4.6. Results RQ5

After exploring the difficult NFRs in specific technologies, we also study the NFRs difficulty over time in the seven categories. Fig. 11 shows a heatmap of NFRs difficulty for the seven specific

Please cite this article as: J. Zou et al., Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis, Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.12.003

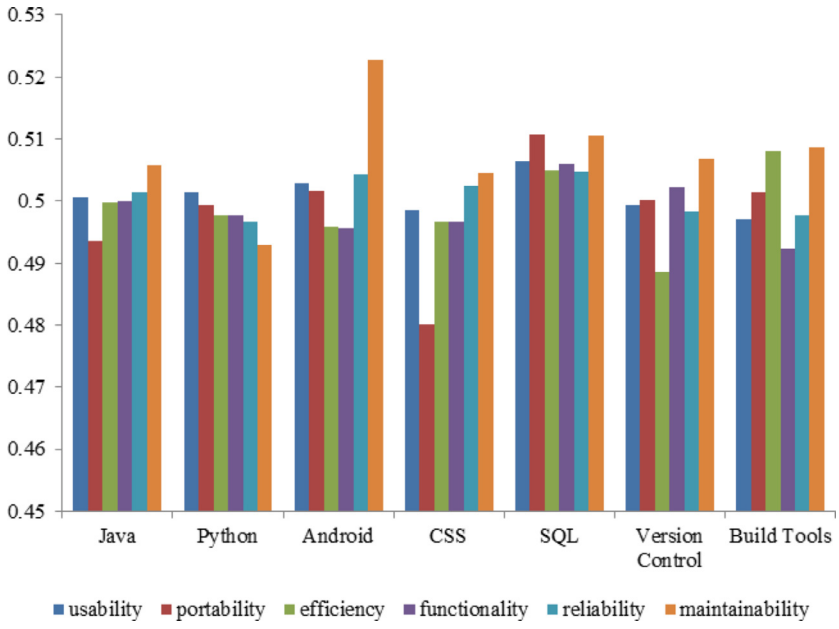


Fig. 10. The difficulty of different NFRs in seven specific technologies.

technologies. From Fig. 11, we see the trends of each NFRs difficulty in every category. We observe that all the NFRs change over time. This indicates that the difficulty is shifted with time. We also see an upward trend of some NFRs difficulty in all categories, such as usability. This signals that we should focus more about usability to address those questions. There are also many NFRs that fluctuate over time, such as the NFRs in Android, maintainability and efficiency in Java. This shows that the NFRs difficulty is also not necessarily strictly increasing or decreasing with time, it might be episodic and responsive to outside stimuli. There are also some categories in which all the six NFRs grow with time in the entire trend, such as Version Control and Python. This indicates that the NFRs in some categories might become more and more difficult. We find that the NFRs difficulty in 2014 in the seven categories is greater than other years. This might be because our data dump is until 2014 so that the percentage of accepted answers and the average number of answers are less than other years.

5. Threats to validity

5.1. Internal validity

When performing the LDA model, we chose to use 20 as the number of topics (K) for each period. There is no absolutely optimal choice for the number of topics in all situations and all datasets [62–64]. The choice is to seek a reasonable value between coarser-grained and finer-grained. To alleviate this threat, we experimented with different values and chose the one that gave the best results by looking at the duplicate words from different topics and the average dominant topic probabilities given by the model. Maybe 20 is not necessarily the optimal choice but it is an appropriate value for our NFRs analysis.

Another threat is the reliability of the NFRs labeling. To mitigate the threat, we evaluate the NFRs labeling on 12 months data using recall rate and precision rate, and the results show it may not be exactly correct, but it is acceptable enough.

When implementing the topic model LDA, LDA models the content of posts as simple text, including the source code. Since there may exist a situation that questions having more source code than simple text, the source code contains words (variable, methods names, etc.) regarding the application domain that may induce LDA to misclassify the concepts. However, in contrast with the number of words in total, there is small number of domain-related words. The analysis provides some form of generality. In the future, we will perform further enhanced investigation by considering removing the source code.

When classifying questions to the seven different categories, we utilized the tags of the posts. It is possible that the tags of posts are omitted by the developer or mislabeled. To mitigate this threat, we manually inspected the posts for the different categories and verified that they contained the proper questions. Another threat is the seven categories picked, maybe there are other more comprehensive categorizations which contain all the discussions in Stack Overflow, but the categorization in the paper is reasonably diverse and representative. We aimed at looking at the Stack Overflow questions of various types to explore the NFRs in the different technologies. The first two (Java, Python) are programming-language specific, the next one (Android) is a specific platform, SQL and CSS are special-purpose programming languages, Version Control and Build Tools are software engineering tools.

When determining if the questions were answered successfully in RQ4, we assumed that the questioners will mark an answer as accepted if the questioners think the answer solves their problem. However, they are not forced to do this and so that it is possible that we did not catch the successfully answered questions accurately.

When dividing the corpus into time-windows, we used the period size of 30 days similar to [7]. We have no evidence to state that 30 days for each period is the most correct partition, but it is an appropriate medium granularity value for our NFRs analysis. The period containing too many posts and therefore aggregating too many words may cause the discrimination between topics is not obvious enough, while the period containing too few data is insufficient for the analysis. The period size of 30 days is smaller than that time span containing too many posts but large enough for there to be adequate posts to analyze.

We tried to minimize the internal threats to validity by using mature tools for extracting data, executing LDA, labeling NFRs and visualization. We used the SAX API to parse and extract data, MAL-

Please cite this article as: J. Zou et al., Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis, Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.12.003

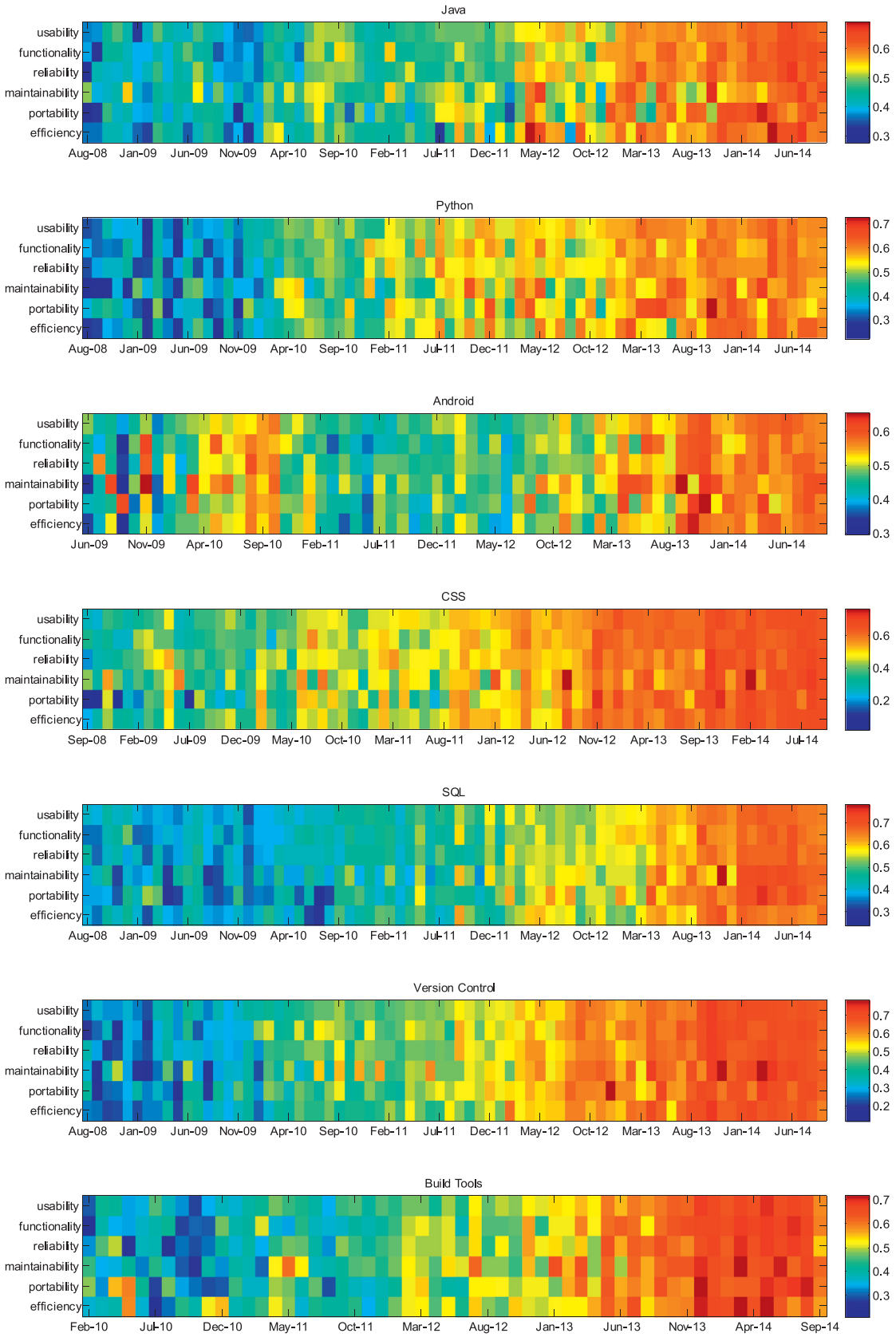


Fig. 11. The difficulty of different NFRs over time in the seven specific technologies.

LET to construct our LDA model, JAVA programming environment for NFRs labeling, Excel and Matlab to visualize figures.

5.2. External validity

One potential threat in this paper is that we used only Stack Overflow data. But as one of the top Q&A sites, it is very large, widely used and popular with tens of thousands of developers worldwide. For this paper, it contains 21.7 million posts and 32.5 million comments. The breadth of Stack Overflow provides some form of generality. In the future, we will perform further enhanced investigation by including other Q&A sites and programming forums.

Another threat is the applying of ISO9126. We have no evidence to state that ISO9126 is the most correct and detailed standard, but it presently constitutes the most widely used software quality model. ISO9126 is "an international standard and thus provides an internationally accepted terminology for software quality" [65]. Thus, we consider it is sufficient for the purposes of this research. In the future, we would consider further enhanced investigation by including other software quality models, such as ISO25010.

5.3. Reliability

When creating the validation corpus, the annotators may exhibit some bias because of the comprehension and subjectivity. To alleviate the threat, we invited four PhD candidates who researched in software engineering to label the topics manually. We also evaluated inter-rater reliability using Fleiss' kappa on a common and small subset.

6. Conclusion

We use a topic model to analyze the NFRs of Stack Overflow data. Our analysis provides an approximation of the comprehension of NFRs through contemporary developers' eyes. It signals the actual wants and needs of the developers. We have explored the NFRs the developers focused on, the NFRs difficulty the developers faced, and the trends of NFRs focus and NFRs difficulty over time. We find that the developers focus most on usability and reliability, while they are less concerned about maintainability and efficiency. We also compared the results of using only posts with the results of using posts and comments, and find that the results of the two settings are almost the same. The most problems that remain unresolved are also related to usability and reliability; they need more help in the usability and reliability areas. And then we analyze the trends of the different NFRs on the posts and the unanswered questions. We observe that the NFRs are changing over time; functionality and reliability increase and the usability always remain hot. Moreover, we investigate the NFRs in specific technologies and analyze the relationships and differences between them. We find that the quality is of similar concern among different technologies. All of those developers focus most on the usability, and less on the portability and maintainability. We also find some NFRs are of more interest with time. After that, we study the NFRs difficulty of each category by applying four metrics. We find that the maintainability is the most difficult NFRs among the six NFRs while the portability is the easiest. We also investigate the NFRs difficulty over time in the seven categories, the results show that all the NFRs difficulties shift with time, the trends indicate that many NFRs difficulty fluctuates over time and the difficulty of usability rises slowly overall.

Acknowledgments

The work described in this paper was partially supported by Chongqing University Postgraduates' Innovation Project (Grant No. CYS15022), the National Natural Science Foundation of China (Grant Nos. 91118005, 61173131), and Changjiang Scholars and Innovative Research Team in University (Grant No. IRT1196).

References

[1] J. Vassileva, Toward social learning environments, Learn. Technol. IEEE Trans. 1 (2008) 199–214.
[2] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent Dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022.
[3] A. Barua, S.W. Thomas, A.E. Hassan, What are developers talking about? An analysis of topics and trends in Stack Overflow, Empirical Softw. Eng. 19 (2014) 619–654.
[4] M. Linares-Vásquez, B. Dit, D. Poshyvanyk, An exploratory analysis of mobile development issues using Stack Overflow, in: Proceedings of the 10th Working Conference on Mining Software Repositories, 2013, pp. 93–96.
[5] C. Rosen, E. Shihab, What are mobile developers asking about? A large scale study using Stack Overflow, Empirical Softw. Eng. (2015) 1–32.
[6] Ø. Hauge, C. Ayala, R. Conradi, Adoption of open source software in software-intensive organizations–A systematic literature review, Inf. Softw. Technol. 52 (2010) 1133–1154.
[7] A. Hindle, M.W. Godfrey, R.C. Holt, What's hot and what's not: windowed developer topic analysis, in: Software Maintenance, 2009. ICSM 2009. IEEE International Conference on, 2009, pp. 339–348.
[8] A. Hindle, N. Ernst, M.W. Godfrey, R.C. Holt, J. Mylopoulos, What's in a name? On the automated topic naming of software maintenance activities, 2010, http://softwareprocess.es/whats-in-a-name.
[9] A. Hindle, C. Bird, T. Zimmermann, N. Nagappan, Relating requirements to implementation via topic analysis: do topics extracted from requirements make sense to managers and developers? in: Software Maintenance (ICSM), 2012 28th IEEE International Conference on, 2012, pp. 243–252.
[10] A. Hindle, N.A. Ernst, M.W. Godfrey, 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, 2011, pp. 163–172.
[11] I. Iso, I. Std, 9126 Software product evaluation–quality characteristics and guidelines for their use, ISO/IEC Standard 9126 (2001).
[12] J. Zou, L. Xu, W. Guo, M. Yan, D. Yang, X. Zhang, Which non-functional requirements do developers focus on? An empirical study on Stack Overflow using topic analysis, in: Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on, 2015, pp. 446–449.
[13] C. Treude, O. Barzilay, M.-A. Storey, How do programmers ask and answer questions on the web?: Nier track, in: Software Engineering (ICSE), 2011 33rd International Conference on, 2011, pp. 804–807.
[14] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, B. Hartmann, Design lessons from the fastest q&a site in the west, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2011, pp. 2857–2866.
[15] M. Asaduzzaman, A.S. Mashiyat, C.K. Roy, K.A. Schneider, Answering questions about unanswered questions of Stack Overflow, in: Proceedings of the 10th Working Conference on Mining Software Repositories, 2013, pp. 97–100.
[16] A. Bosu, C.S. Corley, D. Heaton, D. Chatterji, J.C. Carver, N.A. Kraft, Building reputation in stackoverflow: an empirical investigation, in: Proceedings of the 10th Working Conference on Mining Software Repositories, 2013, pp. 89–92.
[17] K. Bajaj, K. Pattabiraman, A. Mesbah, Mining questions asked by web developers, in: Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 112–121.
[18] Y. Jin, X. Yang, R.G. Kula, E. Choi, K. Inoue, H. Iida, Quick trigger on Stack Overflow: a study of gamification-influenced member tendencies, in: Proceedings of the 12th Working Conference on Mining Software Repositories, 2015, pp. 434–437.
[19] J. Goderie, B.M. Georgsson, B. Van Graafeiland, A. Bacchelli, ETA: estimated time of answer predicting response time in Stack Overflow, in: Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on, 2015, pp. 414–417.
[20] F. Calefato, F. Lanubile, M.C. Marasciulo, N. Novielli, Mining successful answers in Stack Overflow, in: Proceedings of the 12th Working Conference on Mining Software Repositories, 2015, pp. 430–433.
[21] N. Novielli, F. Calefato, F. Lanubile, The challenges of sentiment detection in the social programmer ecosystem, in: Proceedings of the 7th International Workshop on Social Software Engineering, 2015, pp. 33–40.
[22] A. Anderson, D. Huttenlocher, J. Kleinberg, J. Leskovec, Discovering value from community activity on focused question answering sites: a case study of Stack Overflow, in: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2012, pp. 850–858.
[23] R. Slag, M. de Waard, A. Bacchelli, One-day flies on StackOverflow: why the vast majority of stackoverflow users only posts once, in: Proceedings of the 12th Working Conference on Mining Software Repositories, 2015, pp. 458–461.
[24] A. Marder, Stack Overflow badges and user behavior: an econometric approach, in: Proceedings of the 12th Working Conference on Mining Software Repositories, 2015, pp. 450–453.
[25] S.M. Nasehi, J. Sillito, F. Maurer, C. Burns, What makes a good code example?: A study of programming Q&A in StackOverflow, in: Software Maintenance (ICSM), 2012 28th IEEE International Conference on, 2012, pp. 25–34.

Please cite this article as: J. Zou et al., Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis, Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.12.003

[26] V. Honsel, S. Herbold, J. Grabowski, Intuition vs. truth: evaluation of common myths about stackoverflow posts, in: Proceedings of the 12th Working Conference on Mining Software Repositories, 2015, pp. 438–441.
[27] S.A. Chowdhury, A. Hindle, Mining StackOverflow to filter out off-topic IRC discussion, in: Proceedings of the 12th Working Conference on Mining Software Repositories, 2015, pp. 422–425.
[28] H. Li, Z. Xing, X. Peng, W. Zhao, What help do developers seek, when and how? in: 2013 20th Working Conference on Reverse Engineering (WCRE), 2013, pp. 142–151.
[29] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow," Georgia Institute of Technology, Tech. Rep, 2012.
[30] L. Chung, B.A. Nixon, Dealing with non-functional requirements: three experimental studies of a process-oriented approach, in: Proceedings of the 17th international conference on Software engineering, 1995, pp. 25–37.
[31] J. Mylopoulos, L. Chung, B. Nixon, Representing and using nonfunctional requirements: a process-oriented approach, Softw. Eng. IEEE Trans. 18 (1992) 483–497.
[32] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos, Non-functional requirements in software engineering, Springer Sci. Bus. Media 5 (2012).
[33] D. Mairiza, D. Zowghi, N. Nurmuliani, An investigation into the notion of non–functional requirements, in: Proceedings of the 2010 ACM Symposium on Applied Computing, 2010, pp. 311–317.
[34] J. Doerr, D. Kerkow, T. Koenig, T. Olsson, T. Suzuki, Non-functional requirements in industry-three case studies adopting an experience-based NFR method, in: Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on, 2005, pp. 373–382.
[35] M. Glinz, On non-functional requirements, in: in Requirements Engineering Conference, 2007. RE'07. 15th IEEE International, 2007, pp. 21–26.
[36] M. Umar, N.A. Khan, Analyzing non-functional requirements (nfrs) for software development, in: Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on, 2011, pp. 675–678.
[37] M. Galster, E. Bucherer, A taxonomy for identifying and specifying non-functional requirements in service-oriented development, in: Services-Part I, 2008. IEEE Congress on, 2008, pp. 345–352.
[38] D. Ameller, C. Ayala, J. Cabot, X. Franch, How do software architects consider non-functional requirements: an exploratory study, in: Requirements Engineering Conference (RE), 2012 20th IEEE International, 2012, pp. 41–50.
[39] D. Ameller, X. Franch, J. Cabot, Dealing with non-functional requirements in model-driven development, in: Requirements Engineering Conference (RE), 2010 18th IEEE International, 2010, pp. 189–198.
[40] S. Kugele, W. Haberl, M. Tautschnig, M. Wechs, Optimizing automatic deployment using non-functional requirement annotations, in: Leveraging Applications of Formal Methods, Verification and Validation, Springer, 2008, pp. 400–414.
[41] M. Ahmad, N. Belloir, J.-M. Bruel, Modeling and verification of functional and non-functional requirements of ambient self-adaptive systems, J. Syst. Softw. 107 (9) (2015) 50–70.
[42] A. Borg, A. Yong, P. Carlshamre, K. Sandahl, The bad conscience of requirements engineering: an investigation in real-world treatment of non-functional requirements, 2003.
[43] W. Damm, A. Votintseva, A. Metzner, B. Josko, T. Peikenkamp, E. Böde, Boosting re-use of embedded automotive applications through rich components, in: Proceedings of Foundations of Interface Technologies, 2005, 2005.
[44] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc, The detection and classification of non-functional requirements with application to early aspects, in: Requirements Engineering, 14th IEEE International Conference, 2006, pp. 39–48.
[45] J. Eckhardt, D.M. Fernandez, A. Vogelsang, How to specify non-functional requirements to support seamless modeling? A study design and preliminary results, in: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2015, pp. 1–4.
[46] J. Eckhardt, A. Vogelsang, D.M. Fernández, Are non-functional requirements really non-functional? An investigation of non-functional requirements in practice, in: Proceedings of the 38th International Conference on Software Engineering, 2016, pp. 832–842.
[47] M. Yan, X. Zhang, D. Yang, L. Xu, J.D. Kymer, A component recommender for bug reports using discriminative probability latent semantic analysis, Inf. Softw. Technol. 73 (2016) 37–51.
[48] S.K. Lukins, N.A. Kraft, L.H. Etzkorn, Bug localization using latent Dirichlet allocation, Inf. Softw. Technol. 52 (2010) 972–990.
[49] J. Zou, L. Xu, M. Yang, X. Zhang, J. Zeng, and S. Hirokawa, "Automated Duplicate Bug Report Detection using Multi-factor Analysis."
[50] M. Allamanis, C. Sutton, Why, when, and what: analyzing Stack Overflow questions by topic, type, and code, in: Proceedings of the 10th Working Conference on Mining Software Repositories, 2013, pp. 53–56.
[51] A.T. Ying, Mining challenge 2015: comparing and combining different information sources on the Stack Overflow data set, The 12th Working Conference on Mining Software Repositories, 2015.
[52] T.T. Nguyen, T.N. Nguyen, T.M. Phuong, Topic-based defect prediction (nier track), in: Proceedings of the 33rd International Conference on Software Engineering, 2011, pp. 932–935.
[53] Y. Fu, M. Yan, X. Zhang, L. Xu, D. Yang, J.D. Kymer, Automated classification of software change messages by semi-supervised Latent Dirichlet allocation, Inf. Softw. Technol. 57 (2015) 369–377.
[54] S. Thomas, Mining Unstructured Software Repositories using ir Models, 2012.
[55] A.K. McCallum, Mallet: A Machine Learning for Language Toolkit, 2002.
[56] S. Geman, D. Geman, Stochastic relaxation gibbs distributions, and the Bayesian restoration of images, IEEE Trans. Pattern Anal. Mach. Intell. 6 (1984) 721–741.
[57] L.R. Biggers, C. Bocovich, R. Capshaw, B.P. Eddy, L.H. Etzkorn, N.A. Kraft, Configuring latent Dirichlet allocation based feature location, Empirical Softw. Eng. 19 (2014) 465–500.
[58] B.W. Boehm, J.R. Brown, M. Lipow, Quantitative evaluation of software quality, in: Proceedings of the 2nd International Conference on Software Engineering, 1976, pp. 592–605.
[59] J. McCall, Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager, volume 1-3, General Electric, 1977.
[60] F. Schwab, B. Ungar, B. Blondel, J. Buchowski, J. Coe, D. Deinlein, et al., Scoliosis research society—Schwab adult spinal deformity classification: a validation study, Spine 37 (2012) 1077–1082.
[61] E.G. Olds, Distributions of sums of squares of rank differences for small numbers of individuals, Ann. Math. Stat. 9 (1938) 133–148.
[62] S.W. Thomas, B. Adams, A.E. Hassan, D. Blostein, Studying software evolution using topic models, Sci. Comput. Program. 80 (2014) 457–479.
[63] S. Grant, J.R. Cordy, Estimating the optimal number of latent concepts in source code analysis, in: Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on, 2010, pp. 65–74.
[64] H.M. Wallach, I. Murray, R. Salakhutdinov, D. Mimno, Evaluation methods for topic models, in: Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 1105–1112.
[65] J. Boegh, A new standard for quality requirements, IEEE Softw. 25 (2008) 57.

Please cite this article as: J. Zou et al., Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis, Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.12.003