Understanding Developers' Discussions and Perceptions on Non-functional Requirements: The Case of the Spring Ecosystem

ANDERSON OLIVEIRA, PUC-Rio, Brazil
JOÃO CORREIA, PUC-Rio, Brazil
WESLEY K. G. ASSUNÇÃO, North Carolina State University, USA
JULIANA ALVES PEREIRA, PUC-Rio, Brazil
RAFAEL DE MELLO, Federal University of Rio de Janeiro, Brazil
DANIEL COUTINHO, PUC-Rio, Brazil
CAIO BARBOSA, PUC-Rio, Brazil
PAULO LIBÓRIO, PUC-Rio, Brazil
ALESSANDRO GARCIA, PUC-Rio, Brazil

Non-Functional Requirements (NFRs) should be defined in the early stages of the software development process, driving developers to make important design decisions. Neglecting NFRs may lead developers to create systems that are difficult to maintain and do not meet users expectations. Despite their importance, the discussion of NFRs is often ad-hoc and scattered through multiple sources, limiting developers' awareness of NFRs. In that scenario, Pull Request (PR) discussions provide a centralized platform for comprehensive NFR discussions. However, existing studies do not explore this important source of information in open-source software development, which developers widely use to discuss software requirements. In this study, we report an investigation of NFR discussions in PRs of repositories of the Spring ecosystem. We collected, manually curated, and analyzed PR discussions addressing four categories of NFRs: maintainability, security, performance, and robustness. We observed that discussions surrounding these PRs tend to address the introduction of a code change or explain some anomaly regarding a particular NFR. Also, we found that more than 77% of the discussions related to NFRs are triggered in the PR title and/or description, indicating that developers are often provided with information regarding NFRs straightway. To gain additional knowledge from these NFR discussions, our study also analyzed the characteristics and activities of developers who actually discuss and fix NFR issues. In particular, we performed an in-depth analysis of 63 developers that stood out in collaborating with the mapped PRs. To complement this analysis, we conducted a survey with 44 developers to gather their perceptions on NFR discussions. By observing how developers approach NFRs and participate in discussions, we documented the best practices and strategies newcomers can adopt to address NFRs effectively. We also provided a curated dataset of 1,533 PR discussions classified with NFR presence.

CCS Concepts: • **Software and its engineering** → **Software design engineering**; **Requirements analysis**.

Additional Key Words and Phrases: non-functional requirements, open-source systems, social metrics

ACM Reference Format:
Anderson Oliveira, João Correia, Wesley K. G. Assunção, Juliana Alves Pereira, Rafael de Mello, Daniel Coutinho, Caio Barbosa, Paulo Libório, and Alessandro Garcia. 2024. Understanding Developers' Discussions and Perceptions on Non-functional Requirements: The Case of the Spring Ecosystem. In *Proceedings of the*

FSE'24, July 2024, Porto de Galinhas, Brazil
© 2024 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/3643750

32nd ACM Symposium on the Foundations of Software Engineering (FSE '24), July 15–19, 2024, Porto de Galinhas, Brazil. ACM, New York, NY, USA, Article 24, 22 pages. https://doi.org/10.1145/3643750

1 INTRODUCTION

Non-Functional Requirements (NFRs) determine a software system's expected qualities or attributes, such as *security*, *maintainability*, and *performance*. The specification of NFRs on software development is essential to establish the technical constraints in which software systems should run [Casamayor et al. 2010]. Consequently, NFRs support developers in making architectural and design decisions that drive the implementation of a software system [Nuseibeh 2001].

Over the software life cycle, developers are expected to discuss and the impacts of NFRs on the system. Particularly during software maintenance activities, the lack of organized and updated information concerning NFRs has several negative consequences [Saadatmand et al. 2012; Slankas and Williams 2013]. For example, assume developers do not have a proper source of information when reasoning about NFRs. In that case, it can result in misinterpretation of the maintenance tasks, delays in solving issues, or increased risk of introducing new NFR problems (*e.g.*, decreasing *performance*) [Chung and Nixon 1995; Yamashita and Moonen 2012, 2013].

Despite its relevance, the specification of NFRs is commonly neglected or informal [Bhowmik and Do 2019; Chung et al. 2012; Hoskinson 2011; Scacchi 2009]. We observe this behavior in open-source systems (OSSs), where developers typically discuss NFRs on demand during maintenance tasks, using unstructured communication such as mailing lists [de Souza et al. 2005; Noll and Liu 2010]. Since the information about the system NFRs is unavailable or non-organized in a structured and specific/dedicated document, developers need to find alternatives to map and understand the systems' NFRs, such as analyzing scattered textual content [Noll and Liu 2010].

A few studies have investigated alternatives for automating the identification of NFRs on available documentation [Casamayor et al. 2010; Cleland-Huang et al. 2007; Kurtanović and Maalej 2017; Slankas and Williams 2013]. These investigations explore detection approaches ranging from keyword strategies to Natural Language Processing through Machine Learning. However, automatically detected NRFs are typically limited to small datasets of requirement specification documents (*e.g.*, the PROMISE dataset [Baker et al. 2019; Chatterjee et al. 2021; Jindal et al. 2021; Kaur and Kaur 2022]). Furthermore, a set of studies use closed-source systems [Chatterjee et al. 2021; Handa et al. 2022; Mohammed and Alemneh 2021; Tóth and Vidács 2019; Wang et al. 2018], and do not make their artifact available for reproducibility. Yet, to the best of our knowledge, no previous work explored the identification of NFRs in PR discussions.

Pull Requests (PRs) are a common resource employed by developers for discussing functional and non-functional requirements [Gousios et al. 2014, 2015; Jiang et al. 2021; Soares et al. 2015]. A recent study on 900 GitHub projects reveals that more than 54% of projects produce their release notes with PRs (*e.g.*, details of recent changes) [Jiang et al. 2021]. Given the nature of PR discussions, their content can be a valuable source of information on NFRs.

Therefore, in this paper, we report an empirical study aimed at characterizing NFRs that emerged from PR discussions and understanding the attributes of the developers engaged in these discussions. To achieve these aims, we produced a dataset to analyze the NFR discussions, in which we manually classified 1,533 PRs obtained from three Open-Source Systems (OSSs) within the Spring ecosystem [Spring 2022b]. We selected the Spring ecosystem due to the diversity of functionalities and services delivered across its projects [Cosmina et al. 2017]. The Spring ecosystem comprises a large amount of source code supported by a community sharing common concerns, technologies, and human resources. Besides, the Spring community is notably active in discussing code changes through PR discussions. We did an in-depth analysis of three systems from spring and look for a first glimpse of how the NFRs discussions occurred within this community. We focused on four NFR types, namely *maintainability*, *robustness*, *performance*, and *security*, as they were the more prominent ones in the software projects analyzed.

To select the sample of PRs to be manually classified, we applied a pre-classification using keywords associated with to these NFRs in previous studies [Cleland-Huang et al. 2007; Slankas and Williams 2013]. Then, we followed a well-structured procedure for the manual classification of each PR. We classify PR discussions in terms of *(i)* the presence of the NFR type addressed, *(ii)* the location in the PR where the discussions about NFR are triggered, *(iii)* the keywords mentioned in the discussion, and *(iv)* the main message addressing the NFR. This classification allowed us to characterize the PR discussions and the developers discussing NFRs.

Through the analysis of the dataset created, we identified 63 developers that frequently engaged in PRs addressing NFRs. After investigating their characteristics and participation in the discussions, we found that most of these developers play roles related to NFRs (*e.g.*, Security Engineer Senior) in their companies, indicating their expertise in certain NFRs types. This suggests that these NFR discussions can attract attention from experienced developers within the OSS community. In addition, these developers are commonly involved with key tasks in the software system (*e.g.*, commits and reviews). Compared to other developers, their participation in the NFR discussions regarding these tasks is also prevalent. By analyzing the PR discussions classified, we found that the discussions about NFRs typically are triggered by the PR title or description (77%), showing that developers already had an awareness of the NFRs prior to opening the PR. This result suggests that developers consider the NFR not only when initiating PRs but also during various other activities such as testing, maintenance.

To better understand how developers perceive and address NFRs within their systems, we ran a survey with 44 developers. Through this survey, we identified that developers engage in NFR discussions, since they acknowledge their significant influence on software quality. We also observed how the developers address NFRs throughout the entire software development lifecycle, employing multiple methods, including PR discussions and rigorous testing, to ensure the software complies with the systems' NFRs. The participants also highlighted the collaborative aspect of dealing with NFRs during the software evolution, going from meetings between stakeholders to suggestions made by more experienced developers and architects.

In this context, we make the following four contributions:

(1) A characterization of how four common types of NFRs are discussed in PRs. This characterization gives the first glimpse into the nature of NFR discussions in an OSS community, which can help in the development of new technologies for the automatic classification of NFR discussions and warning developers in charge of dealing with NFRs.
(2) We highlight prevalent developers' characteristics while discussing NFRs. By identifying these characteristics, we can support project managers in allocating team members who are ideally more qualified to address a NFR concern [Joblin et al. 2017].
(3) A set of strategies on how to address NFRs. Through the survey applied, we gathered the perceptions of developers regarding NFRs and how they address such requirements in their systems. This information enabled us to formulate an initial set of strategies aimed at assisting developers in effectively addressing NFRs. These strategies include the use of specific technologies and/or practices (*e.g.*, continuous integration) allied with testing and benchmarking to evaluate whether the NFRs defined for the system meet the expectations).
(4) A curated dataset composed of 1,533 PRs addressing *maintainability*, *robustness*, *performance*, and *security* and made publicly available.[1] We also provide a subset of common NFR keywords derived from manual analysis and which refines related knowledge from previous empirical

[1]Available at https://github.com/andersonjso/devs_discussions_perceptions.

studies [Cleland-Huang et al. 2007; Slankas and Williams 2013]. These outcomes also supports further investigations on conceiving, training and validating NFR classification models.

Audience. Project managers shall benefit from our analysis by identifying developers more frequently involved with NFRs. We also provide a documented set of strategies to better address NFRs in software systems based on knowledge gathered through our survey. Moreover, although our analysis is focused on the systems from the Spring ecosystem, the insights gained from our findings can be explored in practice for other ecosystems.

2 BACKGROUND AND RELATED WORK

In this section, we explore *(i)* documentation of NFRs, *(ii)* discussions on OSSs, and *(iii)* developers perceptions on NFRs.

2.1 The Documentation Gap of NFRs

Based on requirements specification, developers are expected to manage and validate the implementation of NFRs through the software development life cycle. However, requirement specification documents usually focus on functional requirements, lacking a sufficient description of the system's NFRs [Cleland-Huang et al. 2007]. In the case of OSSs, even producing a formal specification of the functional requirements is unusual [Bhowmik and Do 2019]. Instead, content addressing the system requirements is often scattered through many documents related to the software system, such as mailing lists, message boards, and ad-hoc discussions [Noll and Liu 2010; Slankas and Williams 2013]. The lack of proper specification and documentation of the system's NFRs is an obstacle to its maintenance and evolution [Robiolo et al. 2019]. For instance, newcomers in charge of evolving systems that lack documentation about NFRs will likely spend considerable time identifying/inferring them. Lack of information may result in an insufficient understanding of the system's NFRs, which may cause design problems [Sousa et al. 2018].

To overcome the problem of insufficient NFR documentation, studies explored different approaches to automatically identify NFRs on the available textual artifacts of the system. Binkhonain and Zao performed a review on machine learning techniques used for the identification and classification of NFRs [Binkhonain and Zhao 2019]. In their reviews, the authors observed a series of limitations on the existing techniques, such as the lack of *(i)* labeled datasets, *(ii)* standard definition, classification, and representation of NFRs, and *(iii)* reporting standards.

Studies have investigated the use of Natural Language Processing (NLP) combined with Machine Learning to automate the identification and classification of NFRs on specification documents. Casamayor *et al.* proposed a semi-supervised learning approach, combining the algorithms Expectation Maximization and Naive Bayes [Casamayor et al. 2010]. The authors empirically evaluated their approach over the PROMISE dataset. The proposed semi-supervised learning approach reached an accuracy rate higher than 75%, surpassing 85% for some NFRs. In addition, Slankas and Willians applied techniques, such as k-NN, Multinomial Naïve Bayes, and Sequential Minimal Optimization (SMO) to identify NFRs from the PROMISE dataset combined with the available documentation [Slankas and Williams 2013]. SMO reached 74% precision and 54.4% recall. However, due to the nature of the documents composing the dataset, most of the NFRs detected address only legal requirements.

The aforementioned studies rely on the PROMISE dataset. However, this dataset comprises only 625 sentences describing requirements specifications (255 functional and 370 non-functional), which may be considered a limited learning scope. Furthermore, this dataset is highly unbalanced. Considering the four NFRs we explore in the OSSs, it only has 74 instances of *maintainability*, 24 of *performance*, 25 of *security*, and none of *robustness*. Therefore, in our study, we could not rely on state-of-the-art techniques to identify the discussions automatically. After creating a model based on the PROMISE dataset, we reached accuracies below 40%. That led us to create our dataset for the purpose of our analysis. We detail this dataset in Section 3.2.

2.2 Open-Source Discussions

OSSs using distributed version control systems tend to use features that support developers in tracking and discussing changes [Gousios et al. 2014]. For example, PRs are one of the most used features in GitHub [Github 2022b]. PRs allow developers to communicate with other team members about changes to be performed [Github 2023]. Once a developer opens a PR, it may discuss the acceptance/rejection of a change with other team members. A PR discussion is composed of *(i)* a title, *(ii)* a description, and *(iii)* comments [Liu et al. 2019]. The PR author creates both the PR title and description. The PR comments result from the system team members' discussion.

Software system discussions represent an important and abundant resource available in OSSs. These discussions may run in the context of reported issues and PRs. On our study we prioritized PR discussions once they often support requirement and design discussions among developers [Gousios et al. 2014]. Even though the issues could also be a source of information on OSS, developers use this mechanism to provide generic problem descriptions. Typically, it leans towards user suggestions rather than specific solutions. Through investigation of software system discussions, we can identify the developers who consistently address NFRs within these discussions. Observing their characteristics, we can understand how the teams address NFRs during the software development. In addition, since these discussions often include the decisions taken to address the NFR, they can serve as documentation to help future developers understand how to properly handle such requirements within the systems.

2.3 Developers' Perception on NFRs

A few studies explore the developers' perception of NFRs [Ameller et al. 2012; Camacho et al. 2016; Zou et al. 2017]. However, these studies lack characterizing how developers discuss NFRs. Zou *et al.* investigated developers' perspectives on NFRs by exploring 21.7 million posts and 32.5 million comments on Stack Overflow [Zou et al. 2017]. For this purpose, the authors applied topic modeling to identify the topics related to NFR discussed in the posts and comments. The authors identified that developers focus more on usability and reliability and are less concerned about efficiency and *maintainability*. They also investigated the difficulty of a topic to be discussed. The authors found that *maintainability* is the most difficult NFR type for developers to discuss.

Ameller *et al.* investigated how software architects perceive and address NFRs in their projects [Ameller et al. 2012]. The authors conducted a semi-structured interview with 20 software architects from academia and industry. They found that the software architects perceive NFRs as key elements for the success of software development. The authors also identified factors that impact how to address the NFRs, such as project size, team composition, and stakeholder involvement.

Camacho *et al.* investigated the perceptions of agile team members regarding the importance of testing NFRs [Camacho et al. 2016]. The authors applied semi-structured interviews with 20 participants from a single company. They observed that several factors, including experience, culture, and awareness, influence the identification of NFRs' testing needs. The authors provide recommendations to address this kind of test better. For instance, code review practices and all roles within the team should work with quality in mind, being aware of the NFRs.

Despite their contributions, these studies have limitations. Zou *et al.* do not investigate the reasons behind the developer's perceptions and focus on questions from Stack Overflow. Hence, it is not clear the experience of the developers involved in the study. Although Ameller *et al.* focus only on software architects, they miss other stakeholders involved in the project. In their study, Camacho *et al.* explore stakeholders of a single company. Our study surveyed more than 40 developers in different positions within their teams and from different companies. In addition, we investigate the different reasons that lead developers to address the NFRs.

Characterizing the developers who deal with NFRs includes identifying, among others, the type of development tasks in which they are more frequently engaged in OSSs, which may indicate their interests and skills. Gonzalez *et al.* investigated the developers' profile based on their productivity (*e.g.*, number of commits) and code quality (*e.g.*, number of refactorings) [González et al. 2021]. They evaluated 77,932 commits from 33 OSSs, clustering 2,460 developers using the k-means algorithm. The authors identified three profiles of developers: *(i)* the cleaner (who documents and fixes issues), *(ii)* the average developer, and *(iii)* the dirty (who introduces complex functions that often need to be refactored). Even though this study followed a more qualitative approach to identifying developer profiles, the authors did not explore whether the developers' profiles deal with NFRs.

3 STUDY DESIGN

To understand PR discussions related to NFRs, we first manually curated a dataset with such discussions. We classified 1,533 PR candidates to address NFRs. Through this classification, we built a dataset to characterize the NFR discussions and identify the developers who discuss NFRs. We selected 63 developers to investigate their characteristics and strategies for dealing with NFRs. Finally, we conducted a survey with developers from private companies to compare their perceptions with the results found in the PRs analysis. The following subsections present the study settings.

3.1 Research Questions

With this study, we have the goal of *understanding developers' discussions and perceptions on NFRs*. Thus, our study is guided by the following research questions (RQs).

RQ1. *What are the characteristics of the discussions in PRs addressing NFRs?*

To answer RQ1, we manually classified NFRs on the PR discussions of three systems from the Spring ecosystem. We focus on analyzing PRs designed to propose changes in the codebase. PRs typically involve several members with different roles, which may lead to a broader view of NFR discussions. For each PR, we first identified the locations (*i.e.*, PR title or PR description) where the contributors address NFRs. With this information, we intend to identify how NFR issues are typically reported in PRs, triggering the discussions. Then, we characterized the NFRs' discussions concerning the topics discussed. We aimed to determine whether it was possible to generalize NFR topics in these discussions. From PR discussions, we could identify the developers most engaged in discussing NFRs (see RQ2).

RQ2. *Who are the developers that engage in NFR discussions?*

By characterizing the developers' expertise and how they contribute to the projects, we aim to understand their role in NFR discussions. We expect to highlight characteristics that developers with specific NFR roles should hold. Once we understand the developers who mainly engaged in NFR discussions in OSSs, we now want to understand developers' perceptions of open-source and closed-source systems. For this purpose, we defined RQ3:

RQ3. *How do developers perceive and address NFRs in their daily work?*

To answer this RQ, we conducted an opinion survey [Linaker et al. 2015] with 44 developers working with multiple closed-source systems. The survey contains questions ranging from the early stages of the software development to its continuous maintenance. Based on the insights gathered, we point out the best strategies developers tend to use when addressing NFRs.

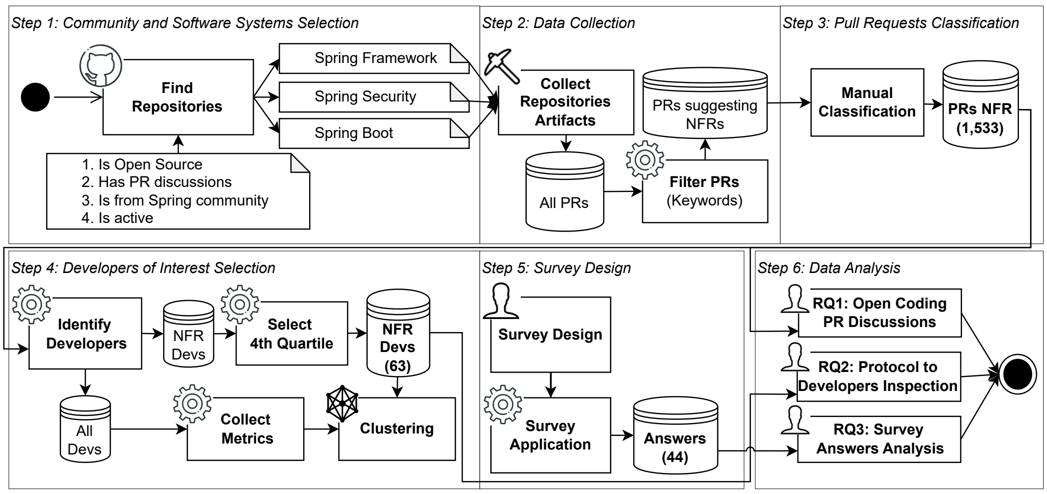

Fig. 1. Workflow of our Study Design.

Table 1. Software Systems from the Spring Ecosystem

System	# Commits	# Pull Requests	Time Span
spring-boot	40,071	5,319	2012-2022
spring-framework	25,522	3,850	2008-2022
spring-security	11,671	1,954	2004-2022

3.2 Study Steps

To answer our RQs, we performed the steps described and presented in Fig. 1.

Step 1. Software Systems Selection: To select suitable systems within the Spring ecosystems for this study, we followed the criteria adopted by Barbosa *et al.* [Barbosa et al. 2020]: *(i)* systems that use PRs as a mechanism to discuss code changes, *(ii)* the system Git software system must have over 1,000 commits, *(iii)* systems that are at least five years old; and *(iv)* systems currently active. Notice that applying such selection criteria is essential for being able to perform reliable conclusions [Kalliamvakou et al. 2016]. We selected the three systems that provided the diversity, considering the entire effort required for the analyses.

Therefore, we selected Spring Security, Spring Framework, and Spring Boot, from a wide and well-known open-source community available at Git: Spring [Spring 2022a]. Table 1 describes the characteristics of each system. We conducted this study over a particular ecosystem to allow us to investigate the dynamics of the NFRs discussions in more depth.

Step 2. Data Collection: We used the GitHub API [Github 2022a] to collect the PR discussions. For each NFR type, we defined a set of keywords that can indicate its incidence in PRs. The set of keywords used in our study is based on previous works [Cleland-Huang et al. 2007; Slankas and Williams 2013]. We collected the keywords on different PR discussion levels (*e.g.*, title, description, and discussion) and the commit messages related to each PR. With this set of keywords, we performed a pre-classification of PR discussions by the NFR type addressed. Based on this pre-classification, we randomly selected a subset of PRs for each NFR type to be manually inspected and classified by experts. The keywords considered in our study are found in Table 2.

Table 2. Non-Functional Requirements and its Keywords

NFRs	Keywords
Maintainability	maintainability, maintenance, reliability, serviceability, accordance, measures, requirements, index, update, release, production, addition, budget, integration, operation, comprehension, readable, readability
Performance	performance, rate, bandwidth, cpu, time, latency, throughput, channel, instruction, response, process, communication, space, memory, storage, peak, compress, uncompress, runtime, perform, execute, dynamic, offset, reduce, response, longer, fast, slow, maximum, capacity, scale, cycle
Robustness	robust, robustness, inputs, error, failure, network, error, reliability, serviceability, fault, tolerance, exception, bug, recover, handl, fail with, crash, unexpect, NPE, null, stack, swallow, reliable
Security	access, author, ensure, data, authentication, security, secure, malicious, prevent, incorrect, harmful, state, exception, vulnerability, vulnerable, malicious, harmful, attack, expose, compromised, authenticator, encrypt, cookie, encrypted, ssl, authenticate, integrity, virus, encryption

Step 3. PRs Classification: First, we defined the NFRs to include in our dataset by identifying the most prominent NFRs within the systems through an analysis of the PR labels. This analysis highlighted four NFRs: *security*, *maintainability*, *robustness*, and *performance*. Since we wanted to characterize the discussions related to NFRs, one of the steps followed was to identify where the mentions of NFRs happen in PRs. Hence, we divided the PR discussion into *(i)* title, *(ii)* description (*i.e.*, first comment), *(iii)* discussion comments, and *(iv)* review comments. We manually classified PRs from each system to identify whether the discussion was related to an NFR. For each PR, the following information was identified:

- The NFR types identified (*i.e.*, security, maintainability, robustness, or performance).
- The sentence (or sentences) that led to each NFR identification.
- The keywords related to each NFR, from single words to short sentences.
- The location of each NFR keyword: title, description, comments, or review comment.
- The participants involved in the PR discussion.

Seven authors performed the manual classification individually. In cases of uncertainty about the classification, a second researcher was consulted. In case of disagreement, the authors discussed it until they reached a consensus. As the output, we have all the PRs tagged with the (set of) NFRs they are discussing. We also used the GitHub API to collect relevant metrics about the developers' participation in NFR discussions. The full list of metrics can be found in our complementary material.[1]

Step 4. Developers of Interest Selection: First, we selected the most engaged developers in NFR discussions. Next, we distributed them into quartiles by the frequency of their participation in NFR discussions. Hence, we defined three participation levels: Low (1st quartile), Average (2nd and 3rd quartiles) and High (4th quartile). We selected the developers in the 4th quartile as the developers of interest since they are the most participative in NFR discussions.

Aiming at also evaluating developers who engaged less frequently in discussions but were more focused on discussing NFRs, we selected another set of developers. For this purpose, we used the following metrics: *(i)* developers who mentioned at least one keyword related to an NFR in at least 25% of their messages within the repository, *(ii)* developers who actively participated in a minimum of three PR discussions, and *(iii)* developers who had less than 50 messages in the repository.

Following these rules, we could find developers who are less active regarding discussions, but their discussions were more focused on NFRs.

To characterize the developers, we computed metrics addressing their background, activities, and code quality (*e.g.*, experience in years). We computed the metrics by gathering raw data through the GitHub API and performing aggregations to compound more complex metrics. The list of metrics can be found in our complementary material.[1]

Next, we applied a clustering algorithm to group the developers based on their metric similarities. We ran a dimensionality reduction in the metrics using PCA [Abdi and Williams 2010]. Then, we conducted the silhouette analysis [Rousseeuw 1987] to estimate the ideal number of clusters that fit our data. Finally, we applied the k-means [MacQueen 1967]. For each software system, we obtained a set of clusters representing similar developers. We combined these steps to select developers of interest to be investigated and analyze their characteristics. We selected 15 developers from the fourth cluster and 48 from the other clusters.

Step 5. Survey Design: We designed a survey to answer RQ3. The survey population comprises software developers with experience dealing with NFRs in their systems. The survey questions and answers are available in our complementary material.[1] The survey questionnaire consists of 19 questions, divided into four blocks, as follows:

- Eight questions to characterize the developers regarding their age, gender, academic experience, experience in software development, role position, and proficiency in programming languages.
- Eight questions aimed at gathering insights into the practices of the developers when dealing with NFRs.
- Two multiple-choice questions, in which we wanted to understand the developers' strategies to address NFRs in their systems. The developers also had the opportunity to add new options if necessary. In addition, we asked them to complement their answer, explaining how these strategies were applied.
- One open question designed to gather insights on the learning and improvement opportunities that the developers experienced when dealing with NFRs.

We conducted the opinion survey in August 2023, extending a 15-day window for responses. We used social media such as Twitter and LinkedIn to publicize the survey. We also publicized with other research groups that we had previously worked with. After this recruitment, the survey questionnaire was answered by 44 developers.

Step 6. Data Analysis: To answer RQ1, on the dataset created, we identified the parts of a PR discussion in which the NFRs appeared (*i.e.*, title, description, messages, and/or review). We did not consider the commit messages since these messages usually appeared as the PR's title and description. To analyze the content of the discussions, we selected a sample of around 15% of the total number of PR discussions in each system, a total of 160 PR discussions for qualitative analysis. We ensured that the discussions selected were equally distributed for each NFR, avoiding bias, and ensured a representative sample for each NFR type.

For this analysis, we followed some steps based on Grounded Theory procedures [Corbin and Strauss 2014]. We applied the *open coding* procedure, which consists of *breakdown, analysis, and categorization* of the data. To develop the coding scheme, we identified the subcategories of each NFR and developed an initial set of codes based on these subcategories (see Section 4.1). Then, we refined the categories by applying the codes to the data and revising as needed. For cases in which the code did not meet any category, we created new ones. The four researchers involved in the open coding received a standard definition of each NFR type to support their activities. A second

Table 3. Distribution of Mentions to the NFRs on Pull Requests Discussions

NFR	Title	Description	Messages	Review	Total
Maintainability	120 (59.11%)	71 (34.97%)	35 (17.24%)	0 (0%)	203
Security	91 (69.46%)	34 (25.95%)	26 (19.84%)	0 (0%)	131
Robustness	58 (51.78%)	45 (40.17%)	29 (25.89%)	2 (1.78%)	112
Performance	38 (58.46%)	32 (49.23%)	13 (20%)	1 (1.53%)	65

author reviewed each code and category created. In case of disagreement, the authors discussed it until they reached a consensus.

To answer RQ2, we investigated 63 developers engaged with NFRs (Step 4). Since the profiles of these developers were not available in a specific document in the projects' repositories, we created a protocol to perform this analysis, in which four authors participated. This protocol includes *(i)* the analysis of the developers' GitHub profile, *(ii)* the identification of other repositories they collaborate, and *(iii)* the manual analysis of their profiles available on the Spring team page [Spring 2022c]. The entire protocol is available in our complementary material.[1] We followed the Grounded Theory procedure to analyze this information, as in RQ1, categorizing and grouping information to find clusters of developers with similar characteristics.

To answer RQ3, we examined the survey answers. We applied descriptive statistics to analyze the numerical and multi-option questions. We also used visualizations (*e.g.*, bar charts, and segmented charts) to observe some tendencies in the results. To evaluate the open questions, we applied qualitative analysis, categorizing the answers to observe the topics most discussed by the developers. Five authors participated in this last analysis, and at least two analyzed each answer. In case of disagreement in the categorization, a third author was asked to participate in a discussion until the team reached a consensus.

4 RESULTS

In this section, we present the results of our study to characterize NFR discussions and understand how the developers perceive and address NFRs in their systems.

4.1 What are the characteristics of the discussions in PRs addressing NFRs?

To characterize the PR discussions related to NFRs, we first inspected where those discussions started and evolved within each PR. For that purpose, in our manual classification (see Section 3.2 - Step 3), we identified which part of the discussion mentioned the NFR keyword (*e.g.*, title, description, discussion messages, and/or review). Table 3 presents the number of times the NFR mention occurred on the PR location. In parenthesis, we show the percentage of mentions compared to all mentions. The last column shows how frequently we observed the NFR in our dataset. For instance, we observed *Maintainability* in 203 PRs. In 120 (59.11%) of them, the mention occurred in the PR title. Notice that, in some cases, the mention appear in multiple locations (*e.g.*, title and description).

By analyzing Table 3, we observe that the PR *title* and the *description* mainly trigger discussions. For all four NFR types, their mention is on the *title* in more than 50% of the instances. When we look at the description, the percentage is lower, even reaching only 25.95% for *Security*. Considering that the *title* and *description* commonly contain information about NFRs, we conducted a more in-depth analysis of their utilization.

Complementary nature of *title* and descriptions. The *title* and *description*, combined, are the pieces of information that the developer must provide to open the PR. Therefore, we also considered when the keyword was in the PR *title* or *description*. We observed that the NFR with the lowest occurrence was *Robustness*, with 77.67% of cases with an NFR mentioned either on the title or *description*. *Maintainability*, *Security*, and *Performance*, reached 83.74%, 84.73%, and 86.15%, respectively. To better understand this relation, consider the example of a PR opened on Spring Framework [Github 2020]. In the PR *title*, the developer shortly described the change performed:

> *"Avoid unnecessary sorting overhead"*

From the mention of the term "overhead", this PR seems related to a change in *Performance*. However, the change is still unclear, in the change description, the developer states:

> *"This PR avoids some unnecessary sorting overhead (e.g. if the collection is too small) for methods that are repeatably called and where collection sizes of <= 1 are fairly common (e.g. for the ProducesRequestCondition)."*

This description clarifies the change performed, allowing other developers to discuss this improvement. In the remaining discussion within this PR, a second developer agreed with the changes and accepted them to be integrated into the system.

Considering the presence of the keywords related to the NFRs in the title and/or description, we observe that in at least 77% of the PRs, the developers were aware of the NFR before opening the PR. This result indicates that developers have NFR concerns during other activities such as tests, maintenance, or external use. We can summarize our findings as follows.

> ***Finding 1:*** *Developers usually mention the NFRs when creating PRs. Thus, developers are aware of and concerned about these NFRs even before opening the PR. This indicates that NFRs are a primary concern during software maintenance and evolution.*

To better understand the content of NFR discussions, we focused on analyzing the titles and descriptions within PR discussions. For that purpose, we randomly selected 160 PRs for qualitative analysis (see Section 3.2 - Step 6). Through open coding, we generated 35 codes and 9 categories, allowing us to understand the content of these titles and descriptions. The list of codes and categories are presented in our complementary material.[1]

NFR problem identification and resolution. Among the codes identified, 25 are associated with the category `NFR Problem Identification` and/or `NFR Problem Resolution`. These two categories describe cases where the developers identify and describe a change related to NFR. The developers either provide a solution themselves or identify the problem and leave it for other developers to solve. Let us consider the example from Spring Boot [Github 2019]. In that case, the developer described a solution to improve the *Performance*. Following, we provide a snippet from what the developer stated:

> *"As described in Issue #16401 there is an optimization opportunity in SpringIterableConfigurationPropertySource when checking for CacheKey equality[...]. This is due to the fact that the internal key inside CacheKey is copied in order to fix PR #13344 and can thus not benefit from a == comparison[...]. The idea of this PR is to introduce a flag that effectively disables the copying of the internal key under certain circumstances[...]. With the applied changes, I see major improvements compared to an M2[2] baseline"*

In the title, this developer described *"optimizing CacheKey handling"*. It is a case where the title and description complement each other. The snippet provided above summarized the PR description. In this example, the developer gave a detailed description of the problem context, where he/she

[2]M2 are the initials to reference a milestone in the system

Table 4. NFR Types and Sub-categories Resulting From the Open-coding on PR Title and Description.

NFR Type	Subcategory
Maintainability	Documentation (28), Code Simplification (18), Feature Enhancement (11), Readability (3), Encapsulation (1), Extensibility (1), Move Component (1), and Setup (1)
Performance	Complexity (6), Concurrency (2), Memory Usage (2), Response Time (1), and Synchronization (1)
Robustness	Error Recoverability (22), Error Representability (9), and Error Scope (4)
Security	Feature Addition (13), Inconsistent Behavior (9), Information Protection (7), and Parameters Customization (1)

found an optimization opportunity on a class in the system. Then, the developer describes how the problem was solved, thus improving *Performance*. Another developer reviewed the code and accepted the PR change in the following discussion. However, this type of long description did not happen so often in our dataset.

NFR-related topics on the PR title and description. We observed multiple codes related to sub-categories concerning the four NFRs, as shown in Table 4. In parentheses, we show the frequency of each subcategory. For each NFR, we subdivided them according to the type of change that was mentioned. We aimed to understand the different topics the title and description address. For instance, we identified 64 cases for *Maintainability*, divided into eight sub-categories: *documentation, code simplification, feature enhancement, readability, encapsulation, extensibility, move component* and *setup*. `Documentation` was the most recurrent topic (28 cases) for PRs that discuss *Maintainability*. In general, those changes are related to the addition of missing documentation, fixing inconsistencies between the documentation and code behavior, typo corrections, and minor improvements. Due to the space limitation, we will only describe one of the sub-categories. The remaining are detailed in our complementary material.[1]

For *Performance*, we identified 12 cases divided into five sub-categories: `complexity`, `concurrency`, `memory usage`, `response time`, and `synchronization`. For this NFR, the most recurrent sub-category was `complexity`, which describes cases where the change increased (or decreased) the computational complexity.

For *Robustness*, we identified 35 codes, divided into three sub-categories related to the NFR: `error recoverability`, `error representability`, and `error scoping`. The most recurrent sub-category for *Robustness* was the `error recoverability` (22 cases). This error happens when there is an improper exception handling and clean-up actions after the system raises an exception.

Finally, for *Security*, we identified 30 cases divided into four sub-categories: `feature addition`, `inconsistent behavior`, `information protection`, and `parameter customization`. While the different categories had similar cases, we can observe that `feature addition` was the most frequent category identified in the coding process. Those cases usually relate to adding a wide range of new security-related features, such as new types of encryption, new authentication methods, etc. With these examples and the data presented in Table 4, we can define our next finding as follows.

Finding 2: *The PR titles and descriptions tend to clearly communicate the intention and the scope of changings addressing NFRs.*

Upon the analyses and findings described above, we can characterize the NFR discussions as being triggered mainly by their title and description and usually discussing the identification and

Table 5. Summary of Developers' Characteristics

Task/ID	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
PRs Opened	15	3	7	1	16	10	3	11	3	0	4	0	7	2	2
PR Comments	115	3	152	60	11	7	39	125	34	49	47	69	18	12	3
PR Reviews	500	36	419	9	2	0	10	595	71	6	4	19	15	24	17
PR Commits	90	10	111	5	17	10	4	86	58	0	9	2	34	4	0

resolution of problems related to the NFR. In addition, these titles/descriptions also encompass multiple sub-categories of the NFRs explored.

By understanding how the PR discussions occur, researchers can explore these characteristics in systems from other communities and domains. Researchers may optimize their efforts based on our findings. For instance, instead of considering the whole discussion to identify a particular NFR discussion, we may consider only the title and description of the PR discussions. It can also avoid noise in the identification process.

In addition, we found in the software systems analyzed that developers may use the PRs titles, descriptions, and code changes as a source of documentation about the system's NFRs. In this way, newcomers can use this content to understand how developers address NFRs during the development process. Having this source of documentation is particularly useful in OSSs, where the NFR documentation is often scattered through multiple sources. Considering the importance of understanding how these discussions occur, it is also important to understand who are the developers engaged in NFR discussions. Therefore, next, we investigate the characteristics of developers who discuss NFRs.

4.2 Who are the developers that most engage in NFR discussions

We first applied the k-means clustering technique for each software system to observe whether developers had similarities regarding their activities. Then, we verified in which cluster each selected developer is classified. For each software system, the clustering results led to four clusters. Notice that although the number of clusters was equal between the three software systems, they didn't exhibit identical characteristics within those clusters. However, we observed similarities between the clusters across repositories, as follows.

- **Newcommers**: It is composed of two clusters, containing around 80% of the total number of developers. Developers in these clusters tend to be newcomers with less experience or other developers who are less active. This is an expected result since the most significant part of contributors is not regular.
- **Active Contributors**: Contains developers who are newbies or recent contributors but highly active. This cluster usually has less than 10% of developers.
- **Experienced Contributors**: They are highly active and experienced contributors on the software system, having top metrics in multiple tasks (*e.g.*, commits, PRs, refactorings, comments, and reviews). Less than 10% of the developers belong to this cluster.

Following, to understand who deals with NFR, we selected 15 developers by identifying their degree of participation in NFR discussions (see Section 3.2 - Step 4). The table respectively presents, for each developer, the number of PRs opened, comments made, reviews, and commits. Table 5 summarizes the characteristics of these 15 developers regarding their main tasks on the software system. For instance, developer #D1 *(i)* opened 15 PRs, *(ii)* made 115 comments, *(iii)* made 500 review comments, and *(iv)* committed 90 times.

We observed that these 15 developers initially analyzed belong to cluster *Experienced Contributors*. They are part of the small group of core contributors active in the software system, not only addressing questions related to NFRs. Therefore, this leads us to our third finding.

***Finding 3:** Developers more actively discussing NFRs are also core contributors having high participation on multiple tasks in the software system (e.g., commits, reviews, and refactorings).*

These findings show the activity of developers on the software system. However, we want to understand what makes these developers discuss NFRs. For that purpose, we analyzed other sources of information available (*e.g.*, GitHub profile and Spring Team page) that support us in understanding what makes developers discuss specific NFRs. Our manual analysis through open coding generated 17 codes and six categories, allowing us to understand these developers. More details about these codes and categories are found in our complementary material.[1]

Developers participation on other projects. In our analysis, we could observe that all 15 developers participated in multiple OSSs. Many of them, such as #D1, have collaborated in more than 10 software systems. For these developers, it is common that most of the systems they participate in are part of the *Spring* ecosystem. Multiple developers have participated in more than 10 software systems from *Spring*. They also tend to participate in repositories related to NFR, the most discussed in the PRs. For instance, #D1 participates in multiple software systems related to *Security*, whereas #D2 participates in several software systems related to documentation, and #D3 participates in software systems related to *Performance*.

Where developers work. We identified where the developers work through a manual analysis. Almost all developers (12 of 15) work for a company that supports the *Spring* ecosystem. One developer works for a company focused on developing browser and mobile games. Two developers do not have their companies identified. Thus, the developers generally work for a company that directly maintains the *Spring* projects or for a company that presumably uses *Spring* products.

These results suggest that most of these developers are knowledgeable about the software system since *(i)* they work for the company that maintains the software system, *(ii)* they describe themselves as engineers for the software system, and *(iii)* they are active in the *Spring* repositories. With these observations, we see why these developers engage more in NFR discussions. Since the companies where they work are closely related to the *Spring*, they have a strong incentive to ensure that the NFRs are properly addressed. Their high activities in tasks such as opening PRs and reviewing others' PRs suggest that they not only have a good understanding of the NFRs of the system, but they are also proactive in addressing them.

Developers' expertise based on their position. Based on the developers' positions in companies, we could infer their expertise on certain NFRs. For instance, the developer #D1 has high participation in multiple tasks related to *Security* in the software system. On the software system team page, we observed that #D1 is *Security Senior Engineer*, a position closely related to the NFR he most discusses. Therefore, due to the nature of this position, we can infer that this developer has high expertise in *Security*. A similar scenario happened with the developer #D2. This developer mainly revised code related to *Maintainability*. By looking at the PR discussions in which this developer participated, it is also mainly related to the documentation of Spring Security. We observed that this developer is *Senior Technical Writer* of his company. Hence, it makes sense for his main contributions to be related to *Maintainability*, with a focus, especially on documentation.

To understand developers who discussed less but were more focused on discussing NFRs, we selected 48 developers to investigate from the three systems. Initially, we selected 20 of each system,

Table 6. Participants Characterization

	Average	**Median**	**Mode**
Age	36.13	33.50	32
Experience in Years	12.06	10	3, 5 and 20
Number of Projects Participated	16.61	8.50	2 and 8

equally distributed considering the four NFRs. However, some developers appeared in more than one of the three systems analyzed, leading us to analyze 48 developers. However, we could not find any specific information about most of them (*e.g.*, they did not have information on their GitHub page).

We can highlight that at least six are open-source enthusiasts and active contributors to multiple open-source projects. Four are main contributors to projects within the *Spring* community. Three of them are closely related to *Security* open-source projects. In this last case, the developers contributed mainly to the Spring Security project. Combined, this information shows similar characteristics from the group of 15 developers early analyzed. These results give us the first glimpse of the characteristics of developers dealing with NFRs on open-source systems. With these aspects identified, we can summarize our next findings as follows:

> ***Finding 4:*** *Developers who discuss NFRs tend to be part of companies closely related to the Spring community or repositories related to the NFR they most discuss, justifying their engagement in addressing NFRs in OSSs.*

By understanding these developers' characteristics, system managers and leaders can allocate specialists on NFRs to specific tasks or positions in the company. In addition, newcomers can benefit from the knowledge that may be acquired by understanding how the more experienced developers deal with NFRs. Moreover, the team managers can also allocate these developers to review decisions regarding the NFRs and avoid future violations of these requirements. Since these violations are symptoms of design problems in the system [Sousa et al. 2017], this concern with the NFRs is a key activity to keep the software system healthy through its evolution.

4.3 How developers perceive and address NFRs in their systems?

To have a broader view of the perception of developers regarding NFRs, we ran a survey with 44 developers from different systems, mainly from closed-source systems. By observing both closed and open-source developers' perspectives, we aim to generalize our findings. Next, we present the results of this survey.

Participants characterization. In Table 6, we provide an initial characterization of participants in your survey. Concerning their experience in years and the number of projects, we can observe that both more experienced and less experienced developers participated in your survey. When we look at their experience in years, we can observe that the developers vary in experience, but through the mean and median, we can observe high levels of experience.

Considering the education level, most participants had a master's degree (17), followed by a bachelor's degree (12) and a PhD degree (8). Regarding the roles in their companies, most of them were software developers (16) and/or software engineers (15). Concerning the programming languages that they were proficient in, Java (27), Python (25), and JavaScript (19) stand out. Notice that the questions about the roles and programming languages were multiple-choice. By observing these demographics,[1] we see a diversity in the profile of the participants.

Why developers discuss NFRs. In our survey, we asked what makes the participants engage in discussions related to NFRs. Most of them answered that their primary motivation is *(i)* the impact on the quality of the software they use (37), *(ii)* the alignment with the software objectives (30), and *(iii)* the potential risks that may arise if the NFR is not addressed (22). These motivations suggest that the participants value the discussion since they perceive the NFRs as key aspects of the software quality. This is indicated once the majority of participants (42 out of 44) stated that they consider NFRs a crucial aspect for the success of the software system. These answers are the first insights into why developers invest time and effort in addressing NFRs.

When developers discuss NFRs. We asked the developers which phase of the software lifecycle they discussed NFRs. Their responses were diverse, mentioning almost all phases: development (31), design (30), and test (25). This resonates with our first finding, showing that developers' concerns with such requirements come from multiple phases of the software, both on closed and open-source systems. from The developer #P11 even mentioned that during the early development phase, the NFRs are easily identified when the team is giving due importance to identifying and analyzing the NFRs.

The developer #P6 mentioned using the PR discussions within their project for revision purposes. In his/her company, PR discussions are used to discuss *Performance* improvements. #P6 also stated the tests being applied at the end of the day to check non-conformities with the *Performance*, which are used to fix problems that could appear in production. #P36 even mentioned a more sophisticated way to address the NFRs. #P36 noted that the project had a spreadsheet showing the system's NFRs, which a software architect filled out, and the NFRs were addressed based on this demand. We have the following finding by understanding why and when developers discuss NFRs in their systems.

Finding 5: *Developers discuss NFRs because they recognize their impact on the software quality, identifying such requirements as key to the success of the software systems. They discuss the NFRs throughout the software's whole lifecycle, using diverse methods to ensure compliance, such as PR discussions and tests.*

How developers perceive and address NFRs. We asked developers about their strategies to guarantee that the NFRs were adequately applied during the software development. Among their answers, two approaches stand out: collaboration with other team members to gather insights regarding the NFRs (35) and review and refine the definitions of NFRs in the early stages of the development process (31).

We observed developers employ specific NFR analysis tools (19). In these cases, we noticed that teams tend to address NFRs reactively. For instance, #P17 explicitly mentioned that the NFRs were addressed only when specific metrics were below acceptable. In this example, automated tools were used to ensure that the NFR was being met (*e.g.*, use of SonarQube for minimum coverage of automated tests). In addition, #P17 mentioned that their approach was proactive only if the NFR was a requirement defined early by the team of requirements. That reactive approach appeared when they expressed no defined process to deal with the NFRs. Altogether, developers cited multiple approaches to ensure the system meets NFRs: pen tests for *Security*, continuous integration using Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and stress testing for *Performance*.

Other developers mentioned specific strategies to address the NFRs. #P13 mentioned that the team uses developers' previous experiences to define particular NFRs for the system, and then they are reevaluated if necessary during the review phases. #P12 stated that they first looked for NFR documentation available. However, in cases where this was not available, they asked for help from more experienced team members. If there is still no solution, a meeting is scheduled between the stakeholders involved. #P12 even mentioned that if it reaches the point of needing to do this meeting, it was proof of a failure in requirements collection and analysis. Indeed, the lack of proper documentation was the most mentioned in our questions to understand the challenges in addressing NFRs (20 cases). We can resume these results in the following finding.

Finding 6: *Developers primarily address NFRs through collaborative efforts, often resorting to reactive approaches, focusing on NFRs only when automated tools indicate possible problems. This happens because the missing documentation is one of the most significant challenges.*

Which developers address NFRs. We asked the developers whether there is someone specifically dealing with the NFRs in the system. Most developers (26) answered that there is no specific role to handle such requirements. Nine developers mentioned that the entire team is responsible for the NFRs, depending on what is needed. For instance, #P4 stated that the most experienced architects and/or developers were most concerned with the NFRs.

Since, in multiple cases, no developer focused on only solving NFR problems, some practices are followed in the companies to ensure that the NFRs are being adopted. #P30 stated that before joining the company, the team members were trained to understand and keep the NFRs *Security* and *Maintainability* of all activities carried out within it. #P33 also mentioned that within the company, they had specialists in certain NFRs, such as *Security*. However, they are not exclusive to only one project. Instead, they work on all of the company's projects. Compared to our analysis of the open-source systems, our fourth finding stated that the developers who mainly discuss the NFRs are also part of multiple projects within the Spring community. Hence, we can note a similarity in who addresses the NFRs within closed and open-source systems.

Strategies to address NFRs. We gathered information on the best strategies to address NFRs within the developers' systems. We categorized these strategies based on their answers, as follows:

- **Prioritization and planning:** NFRs should be treated with as much priority as other requirements. They should be planned in advance and reviewed throughout the project.
- **Identification and discussion:** NFRs should be identified and discussed early in the development process, ideally in the design phase. During the evolution of the software, these NFRs should be revisited if necessary and discussed with stakeholders the possible changes that will be done to keep the NFR adequate.
- **Used of technologies allied with testing:** The adequacy of the NFR can be verified through technologies already approved by the market, where their non-functional requirements satisfy the project's complexity. In addition, the processes of *continuous integration* and *continuous delivery* (CI/CD) can help in the identification of such problems.
- **Benchmarks:** Using benchmarks to simulate the behavior of a piece of code or algorithm under different conditions is recommendable, enabling the review and refactoring of the code when it is not meeting the project-specified NFRs.
- **Documentation of best practices:** By keeping the NFRs well-documented, developers will have a starting point to address an NFR problem when it appears. In addition, it is good to

have these best practices on handling NFRs documented since each system may have its own particularity.

- **Long-Term mindset:** Properly addressing NFRs will enhance the software's lifetime. To guarantee this, a system *(i)* should have a good user experience, *(ii)* should be designed to scale, and *(iii)* should be easy to maintain by future developers. These characteristics rely on NFRs such as *performance*, *robustness*, *security*, and *maintainability*.

By investigating how developers perceive and handle NFRs in open-source and closed-source systems, we obtained initial insights into their discussions regarding these system requirements. Through an analysis of their perspectives and typical tasks, we compiled a documented overview of best practices for addressing NFRs to ensure the long-term health of software systems. As one survey participant aptly said: *"Paying attention to NFRs can mean the difference between the success or failure of a software project"*.

5 THREATS TO VALIDITY

This section discusses the threats to the *internal* and *external* validity of our study as follows.

5.1 Threats to Internal Validity

Labeling process. The procedure adopted for the manual identification of NFRs in a PR can be posed as a threat. We adopted a well-structured process to overcome such a threat to the study's validity. Besides all authors being experienced with the NFRs types investigated, we also provided complementary material with details about the NFRs to avoid misclassification. In addition, our qualitative analysis ensured that the author evaluated a PR classified by other authors. Finally, when there was any doubt about the classification, a second author also classified the PR to ensure the reliability of the classification.

Clustering algorithm. Regarding clustering, the k-means [MacQueen 1967] algorithm expected an explicit indication of the number of clusters for grouping the data. We mitigate that threat by applying the Silhouette analysis [Rousseeuw 1987] that enables us to determine the most appropriate number of clusters for our data. Furthermore, the set of metrics supplied to the clustering algorithm to group developers could be a limitation. For this reason, we manually reviewed the metrics in the feature engineering process to ensure they represent the proper dimensions that characterize the developer's tasks in the repository.

5.2 Threats to External Validity

System ecosystem generalization. We selected three systems for our analysis that might be a threat. However, our selection was based on a set of criteria (Section 3.2) to ensure that we are selecting project repositories with proper characteristics for our study. In this way, we decided to focus on OSSs from the same ecosystem to perform an in-depth analysis and understand this scenario in specific. We decided to not expand to other ecosystems since how each ecosystem/community discusses NFRs may be different, which could introduce noise in our results. Therefore, we first focused on understanding how NFRs are discussed in a big ecosystem with well-defined policies for PR discussions. Even though the generalization of the study findings is limited to the Spring ecosystem, we provided a detailed research methodology, allowing the replication of the study steps to investigate other software (eco)systems.

Developers' generalization. The number of developers analyzed in *RQ2* can be another threat. However, since our goal was to analyze only the developers dealing with NFRs, we focused on two steps to select them. First, we divided them into quartiles regarding software system activity and their presence in NFR-related discussions. Then, we first selected the developers who were present on the 4th quartile, meaning they were highly active on the software system and regarding the NFRs. Second, we selected developers from the other quartiles but with a higher focus on NFR in the discussions. Finally, we applied the survey to complement the information gathered from these first developers.

NFRs' generalization. The selection of the four NFRs for this study may be posed as a threat, since these NFRs are representative only of the systems that we analyzed. However, we focused mainly on the most discussed NFRs within these systems (Section 3.2 - Step 3). This way, we could perform a more in-depth analysis of such requirements. Our goal was to understand how an NFR is discussed, regardless of its type. Furthermore, we provided detailed research methodology on how to select such NFRs on software systems, which allows a replication of these steps for the identification of NFRs prominent on specific systems.

6 CONCLUSION

To understand how developers discuss NFRs, we first characterized these discussions on three systems from the Spring ecosystem. For that purpose, seven authors of this paper performed a manual classification of 1,533 PR discussions. We considered four NFRs for this classification: *maintainability*, *security*, *performance*, and *robustness*, which were the NFRs most present on the software systems analyzed. Through this classification, we built a dataset of PR discussions regarding the presence of NFRs. With this dataset, we characterized the NFR discussions and identified the developers discussing these requirements.

We observed that in more than 77% of the PRs, discussions related to NFRs are triggered in the title and description, both provided by the developer opening the PR. That allow us understand that the developer who opens the PR knows the NFR that should be discussed. These discussions' content is mainly related to introducing and resolving NFR problems. When we investigated the developers discussing NFRs, we observed that they have high participation in multiple tasks in the software system (*e.g.*, commits and reviews). They also usually have a role in the company that they work for, closely related to a specific NFR (*e.g.*, Spring Security Senior Engineer).

As a contribution, we provide the dataset created by our manual classification, which can be used to develop new and improved automated classification mechanisms for NFRs. The characterization we provided of PR discussions can be used as a starting point for developing tools for automatically detecting NFRs. Furthermore, the mapping of developers who discuss NFR can be used by system managers who want to understand specialists' characteristics in certain NFRs. Hence, they can allocate their team based on these skills. In future studies, we plan to explore the impact the identification of NFRs on PR discussions can have on the system's design and expand our analyses with more ecosystems.

ACKNOWLEDGMENTS

This work is partially funded by CNPq (140185/2020-8, 140770/2021-6, 141180/2021-8, 434969/2018-4, 140771/2021-2, and 312149/2016-6), CAPES/Procad (175956), CAPES/Proex (88887.373933/2019-00), FAPERJ (200773/2019, and 010002285/2019), FAPERJ/PDR-10 program (202073/2020), CAPES (88887.904409/2023-00), and by the Instituto Estadual de Engenharia e Arquitetura (IEEA), Secretaria Estadual de Infraestrutura do Estado do Rio de Janeiro (001/2021).

REFERENCES

Hervé Abdi and Lynne J Williams. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics* 2, 4 (2010), 433–459. https://doi.org/10.1002/wics.1246

David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. 2012. How do software architects consider non-functional requirements: An exploratory study. In *2012 20th IEEE international requirements engineering conference (RE)*. IEEE, 41–50. https://doi.org/10.1109/RE.2012.6345838

Cody Baker, Lin Deng, Suranjan Chakraborty, and Josh Dehlinger. 2019. Automatic Multi-class Non-Functional Software Requirements Classification Using Neural Networks. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2. 610–615. https://doi.org/10.1109/COMPSAC.2019.10275

Caio Barbosa, Anderson Uchôa, Filipe Falcao, Daniel Coutinho, Hyago Brito, Guilherme Amaral, Alessandro Garcia, Baldoino Fonseca, Marcio Ribeiro, Vinicius Soares, and Leonardo Sousa. 2020. Revealing the Social Aspects of Design Decay: A Retrospective Study of Pull Requests. In *34th SBES*. 1–10. https://doi.org/10.1145/3422392.3422443

Tanmay Bhowmik and Anh Quoc Do. 2019. Refinement and resolution of just-in-time requirements in open source software and a closer look into non-functional requirements. *Journal of Industrial Information Integration* 14 (2019), 24–33. https://doi.org/10.1016/j.jii.2018.03.001

Manal Binkhonain and Liping Zhao. 2019. A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications: X* 1 (2019), 100001. https://doi.org/10.1016/j.eswax.2019.100001

Cristina Rosa Camacho, Sabrina Marczak, and Daniela S Cruzes. 2016. Agile team members perceptions on non-functional testing: influencing factors from an empirical study. In *2016 11th international conference on availability, reliability and security (ARES)*. IEEE, 582–589. https://doi.org/10.1109/ARES.2016.98

Agustin Casamayor, Daniela Godoy, and Marcelo Campo. 2010. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Information and Software Technology* 52, 4 (2010), 436–445. https://doi.org/10.1016/j.infsof.2009.10.010

Ranit Chatterjee, Abdul Ahmed, Preethu Rose Anish, Brijendra Suman, Prashant Lawhatre, and Smita Ghaisas. 2021. A Pipeline for Automating Labeling to Prediction in Classification of NFRs. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*. 323–323. https://doi.org/10.1109/RE51729.2021.00036

Lawrence Chung and Brian A. Nixon. 1995. Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach. In *17th International Conference on Software Engineering* (Seattle, Washington, USA) *(ICSE '95)*. Association for Computing Machinery, New York, NY, USA, 25–37. https://doi.org/10.1145/225014.225017

Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. 2012. *Non-functional requirements in software engineering*. Vol. 5. Springer Science & Business Media. https://doi.org/10.1007/978-1-4615-5269-7

Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. 2007. Automated classification of non-functional requirements. *Requirements engineering* 12, 2 (2007), 103–120. https://doi.org/10.1007/s00766-007-0045-1

Juliet Corbin and Anselm Strauss. 2014. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage publications.

Iuliana Cosmina, Rob Harrop, Chris Schaefer, Clarence Ho, Iuliana Cosmina, Rob Harrop, Chris Schaefer, and Clarence Ho. 2017. Introducing Spring. *Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools* (2017), 1–18.

Sergio Cozzetti B de Souza, Nicolas Anquetil, and Káthia M de Oliveira. 2005. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*. 68–75. https://doi.org/10.1145/1085313.1085331

Github. 2019. Optimize CacheKey handling in SpringIterableConfigurationPropertySource by dreis2211 · Pull Request #16717 · spring-projects/spring-boot · GitHub. https://github.com/spring-projects/spring-boot/pull/16717. (Accessed on 10/21/2022).

Github. 2020. Avoid unnecessary sorting overhead by dreis2211 · Pull Request #24617 · spring-projects/spring-framework · GitHub. https://github.com/spring-projects/spring-framework/pull/24617. (Accessed on 10/21/2022).

Github. 2022a. GitHub REST API - GitHub Docs. https://docs.github.com/en/rest. (Accessed on 10/21/2022).

Github. 2022b. GitHub: Where the world builds software · GitHub. https://github.com/. (Accessed on 10/21/2022).

Github. 2023. Available: https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests.

Cristina Aguilera González, Laia Albors Zumel, Jesús Antonanzas Acero, Valentina Lenarduzzi, Silverio Martíncz-Fernández, and Sonia Rabanaque Rodríguez. 2021. A preliminary investigation of developer profiles based on their activities and code quality: Who does what?. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 938–945. https://doi.org/10.1109/QRS54544.2021.00103

Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering*. 345–355. https://doi.org/10.1145/2568225.2568260

Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: The integrator's perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 358–368. https://doi.org/10.1109/ICSE.2015.55

Naina Handa, Anil Sharma, and Amardeep Gupta. 2022. Framework for prediction and classification of non functional requirements: a novel vision. *Cluster Computing* 25, 2 (2022), 1155–1173. https://doi.org/10.1007/s10586-021-03484-0

C. Hoskinson. 2011. Available: http://www.politico.com/news/stories/0611/58051.html.

Huaxi Jiang, Jie Zhu, Li Yang, Geng Liang, and Chun Zuo. 2021. Deeprelease: Language-agnostic release notes generation from pull requests of open-source software. In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 101–110. https://doi.org/10.1109/APSEC53868.2021.00018

Rajni Jindal, Ruchika Malhotra, Abha Jain, and Ankita Bansal. 2021. Mining Non-Functional Requirements using Machine Learning Techniques. *e-Informatica Software Engineering Journal* 15, 1 (2021). https://doi.org/10.37190/e-inf

Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 164–174. https://doi.org/10.1109/ICSE.2017.23

Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071. https://doi.org/10.1007/s10664-015-9393-5

Kamaljit Kaur and Parminder Kaur. 2022. SABDM: A self-attention based bidirectional-RNN deep model for requirements classification. *Journal of Software: Evolution and Process* (2022), e2430. https://doi.org/10.1002/smr.2430

Zijad Kurtanović and Walid Maalej. 2017. Automatically classifying functional and non-functional requirements using supervised machine learning. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. Ieee, 490–495. https://doi.org/10.1109/RE.2017.82

Johan Linaker, Sardar Muhammad Sulaman, Martin Höst, and Rafael Maiani de Mello. 2015. Guidelines for conducting surveys in software engineering v. 1.1. (2015). https://doi.org/10.1007/978-3-030-32489-6_3

Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2019. Automatic generation of pull request descriptions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 176–188. https://doi.org/10.1109/ASE.2019.00026

J MacQueen. 1967. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*. 281–297.

Esmael Mohammed and Esubalew Alemneh. 2021. Identification of Architecturally Significant Non-Functional Requirement. In *2021 International Conference on Information and Communication Technology for Development for Africa (ICT4DA)*. IEEE, 24–29. https://doi.org/10.1109/ICT4DA53266.2021.9672235

John Noll and Wei-Ming Liu. 2010. Requirements elicitation in open source software development: a case study. In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. 35–40. https://doi.org/10.1145/1833272.1833279

Bashar Nuseibeh. 2001. Weaving together requirements and architectures. *Computer* 34, 3 (2001), 115–119. https://doi.org/10.1109/2.910904

Gabriela Robiolo, Ezequiel Scott, Santiago Matalonga, and Michael Felderer. 2019. Technical debt and waste in non-functional requirements documentation: An exploratory study. In *Product-Focused Software Process Improvement: 20th International Conference, PROFES 2019, Barcelona, Spain, November 27–29, 2019, Proceedings 20*. Springer, 220–235. https://doi.org/10.1007/978-3-030-35333-9_16

Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65. https://doi.org/0.1016/0377-0427(87)90125-7

Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. 2012. Toward model-based trade-off analysis of non-functional requirements. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 142–149. https://doi.org/10.1109/SEAA.2012.23

Walt Scacchi. 2009. Understanding requirements for open source software. In *Design Requirements Engineering: A Ten-Year Perspective: Design Requirements Workshop, Cleveland, OH, USA, June 3-6, 2007, Revised and Invited Papers*. Springer, 467–494.

John Slankas and Laurie Williams. 2013. Automated extraction of non-functional requirements in available documentation. In *2013 1st International workshop on natural language analysis in software engineering (NaturaLiSE)*. IEEE, 9–16.

Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance factors of pull requests in open-source projects. In *30th Annual ACM Symposium on Applied Computing*. 1541–1546. https://doi.org/10.1145/2695664.2695856

L. Sousa, A. Oliveira, W. Oizumi, S. Barbosa, A. Garcia, J. Lee, M. Kalinowski, R. de Mello, B. Fonseca, R. Oliveira, C. Lucena, and R. Paes. 2018. Identifying Design Problems in the Source Code: A Grounded Theory. In *40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. ACM, New York, NY, USA, 921–931. https://doi.org/10.1145/3180155.3180239

Leonardo Sousa, Roberto Oliveira, Alessandro Garcia, Jaejoon Lee, Tayana Conte, Willian Oizumi, Rafael de Mello, Adriana Lopes, Natasha Valentim, Edson Oliveira, and Carlos Lucena. 2017. How Do Software Developers Identify Design

Problems?: A Qualitative Analysis. In *Proceedings of 31st Brazilian Symposium on Software Engineering* (Fortaleza, Ceará, Brazil) *(SBES'17)*. 12 pages. https://doi.org/10.1145/3131151.3131168

Spring. 2022a. Spring | Community. https://spring.io/community. (Accessed on 10/21/2022).

Spring. 2022b. Spring | Home. https://spring.io/. (Accessed on 10/21/2022).

Spring. 2022c. Spring | Team. https://spring.io/team. (Accessed on 10/21/2022).

László Tóth and László Vidács. 2019. Comparative Study of The Performance of Various Classifiers in Labeling Non-Functional Requirements. *Information Technology and Control* 48, 3 (2019), 432–445. https://doi.org/10.5755/j01.itc.48.3.21973

Chong Wang, Fan Zhang, Peng Liang, Maya Daneva, and Marten Van Sinderen. 2018. Can app changelogs improve requirements classification from app reviews? an exploratory study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–4. https://doi.org/10.1145/3239235.3267428

A Yamashita and L Moonen. 2012. Do code smells reflect important maintainability aspects?. In *ICSM12*. 306–315. https://doi.org/10.1109/ICSM.2012.6405287

A. Yamashita and L. Moonen. 2013. Do developers care about code smells? An exploratory survey. In *20th Working Conference on Reverse Engineering*. 242–251. https://doi.org/10.1109/WCRE.2013.6671299

Jie Zou, Ling Xu, Mengning Yang, Xiaohong Zhang, and Dan Yang. 2017. Towards comprehending the non-functional requirements through developers' eyes: An exploration of stack overflow using topic analysis. *Information and Software Technology* 84 (2017), 19–32. https://doi.org/10.1016/j.infsof.2016.12.003

Received 2023-09-28; accepted 2024-01-23