Tag that issue: Applying API-domain labels in issue tracking systems

Fabio Santos . Joseph Vargovich . Bianca Trinkenreich . Italo Santos . Jacob Penney . Ricardo Britto . João Felipe Pimentel . Igor Wiese . Igor Steinmacher . Anita Sarma . Marco A. Gerosa

Abstract Labeling issues with the skills required to complete them can help contributors to choose tasks in Open Source Software projects. However, manually labeling issues is time-consuming and error-prone, and current automated approaches are mostly limited to classifying issues as bugs/non-bugs. We investigate the feasibility and relevance of automatically labeling issues with what we call "API-domains," which are high-level categories of APIs. Therefore, we posit that the APIs used in the source code affected by an issue can be a proxy for the type of skills (e.g., DB, security, UI) needed to work on the issue. We ran a user study (n=74) to assess API-domain labels' relevancy to potential contributors, leveraged the issues' descriptions and the project history to build prediction models, and validated the predictions with contributors (n=20) of the projects. Our results show that (i) newcomers to the project consider API-domain labels useful in choosing tasks, (ii) labels can be predicted with a precision of 84% and a recall of 78.6% on average, (iii) the results of the predictions reached up to 71.3% in precision and 52.5% in recall when training with a project and testing in another (transfer learning), and

Fabio Santos, Joseph Vargovich, Bianca Trinkenreich, Italo Santos, Jacob Penney, João Felipe Pimentel, Igor Steinmacher, Marco A. Gerosa
Northern Arizona Unversity
E-mail: fabio_santos@nau.edu, joseph_vargovich@nau.edu, bianca_trinkenreich@nau.edu, italo_santos@nau.edu, jacob_penney@nau.edu, joao.pimentel@nau.edu, igor.steinmacher@nau.edu, marco.gerosa@nau.edu

Igor Wiese,
Universidade Tecnológica Federal do Paraná
E-mail: igor@utfpr.edu.br

Anita Sarma
Oregon State University
E-mail: anita.sarma@oregonstate.edu

Ricardo Britto
Ericsson - Blekinge Institute of Technology
E-mail: ricardo.britto@ericsson.com

(iv) project contributors consider most of the predictions helpful in identifying needed skills. These findings suggest our approach can be applied in practice to automatically label issues, assisting developers in finding tasks that better match their skills.

Keywords API identification · Labelling · Tagging · Skills · Multi-Label Classification · Mining Software Repositories

1 Introduction

Choosing a task to contribute to in Open Source Software (OSS) projects can be challenging [1, 2, 3, 4, 5]. Open tasks are publically reported in issue trackers, but since issues vary in complexity and required skills, contributors may find it difficult to select an appropriate task to undertake, especially when the contributors are new in the projects [6, 7, 8, 9]. Adding labels to the issues (a.k.a. "tasks," "tickets," and "bug reports") is an effective way to help new contributors choose where to focus their efforts [10]. The labeling strategy supports a variety of contributors, including newcomers (new contributors), frequent contributors, and maintainers, as they have similar perceptions of the importance of this strategy [11]. Developers are newcomers each time they start a new project, no matter their previous experience. Nevertheless, community managers and project maintainers find manually labeling issues challenging and time-consuming [12].

We posit that the underlying APIs (the libraries required and imported into the source code) can be parsed to indicate skills required to work on an issue. APIs are defined as "a set of functions and procedures that enable the creation of applications that access the resources or data of an operating system, application or other services" [13]. If the contributors know what types of APIs are used in the code to solve the issue, they could choose tasks that better match their skills or involve skills they want to learn. We leverage the idea that APIs encapsulate modules with specific purposes (e.g., cryptography, database access, logging) and abstract the details from the implementation. In this study, we focus on API-domain labels: high-level labels designating categories of APIs such as "UI," "Security," and "Test," which may relate to skills needed to work on the issues.

This paper extends our prior work [14], in which we conducted a case study with a single project to investigate the feasibility of automatically labeling issues with API-domain labels. After running the first predictions with the case study, we observed that the number of dataset rows dropped significantly because of the lack of information about linked issues and PRs. With this in mind and to improve generalization, we looked for feasible ways to increase the datasets when a project is seriously affected by the dataset size after discarding issues not linked with a PR. In addition, it is sometimes impossible to access the source due to confidentiality in industry projects. Pursuing this reasoning, we sought ways to keep predicting the API-domain labels even when no training data is available by transferring the learning. Therefore, we believe

the API-domain labels should be even more helpful if they can be applied in many open-source projects or industry projects despite their source code's dataset size or availability. We extend the work by (1) expanding our study to five projects with diverse programming languages, vocabularies (natural languages), and issue track systems (ITS), (2) adding the BERT technique to our approach, (3) extending the qualitative analysis, (4) exploring the ability to transfer learning across projects, and (5) evaluating the API-domain labels with developers who solved the issues.

We answer the following research questions:

RQ.1: How relevant are the API-domain labels to new contributors?
RQ.2: To what extent can we automatically attribute API-domain labels to issues?
RQ.2.1: To what extent can we automatically attribute API-domain labels to issues using data from the project?
RQ.2.2: To what extent can we automatically attribute API-domain labels to issues using data from other projects?
RQ.2.3: To what extent can we automatically attribute API-domain labels to issues using transfer learning?
RQ.3: How well do the API-domain labels match the skills needed to solve an issue?

This paper studies the relevance of this labeling strategy to new contributors (RQ.1). We created models and evaluated their performance. Usually, machine learning approaches train and test data with the same project (RQ.2.1). However, when the existing data is not enough to create a prediction model with the expected performance, one may consider enlarging the dataset to include other projects (RQ.2.2). In addition, with the total absence of historical data for training in a target project, one can use a pre-trained dataset in the same domain (source project) to run predictions in the target project [15] (RQ.2.3). Therefore, we also conducted transfer learning studies. Finally, the developers' opinions about the predictions were studied to determine whether the API-domain labels adequately indicate skills and help newcomers choose their tasks (RQ.3).

Our contribution includes (1) how newcomers see the relevance of the API-domain labels; (2) a new semi-automated API classification process; (3) a mechanism to predict skills needed for projects coded in diverse programming languages (C, C#, and Java), with issues in Portuguese and English; and (4) the validation of the API-domain labels with developers.

2 Related Work

Organizing issues involve some labeling efforts. Labeling is important for describing features and making it easier and faster to understand and search through software artifacts [11]. However, manually labeling software artifacts

can be difficult and time-consuming. Thus, some approaches have been proposed for automatically labeling software projects [16] and dependencies [17]. While these approaches demonstrate the possibility of labeling software artifacts, they work at a higher level of classifying the whole project. In contrast, our approach classifies minor software artifacts (i.e., issues and APIs).

Approaches have also been proposed for labeling other software artifacts, such as questions from Stack Overflow [18, 19, 20]. Xia et al. [18] recommend tags for questions based on the similarity with previous questions. Those approaches are restricted to using only the existing tags and do not work with issue-tracking systems or APIs. Uddin and Khomh [20] and Lin et al. [19] label opinions from users about APIs. Despite their focus on APIs, their goal is to support the developers' decisions to adopt a new API. In this work, we have the opposite goal. Given that a project already has APIs in different domains, our goal is to enable developers to find tasks that include APIs with which they are more familiar.

While many approaches are designed to label issues, most of them only try to distinguish bug reports from non-bug reports [21, 22, 23, 24, 25, 26]. Few approaches can classify according to other labels [27, 28, 29]. Among them, Izadi et al. [28] and Wang et al. [29] use the text classification algorithm BERT [30] for multiple labels, which we also use. Despite their ability to classify into distinct labels, such approaches only use pre-existing labels for classification. Instead of using predefined labels extracted from the existing issues or provided by default on GitHub, our approach define labels based on API domains. This kind of labeling helps to guide new contributors toward what to contribute [31, 10], which can be a daunting task without guidance [2].

With a similar goal to support new contributors, social coding platforms like GitHub[1] encourage projects to label issues[2] that are easy for new contributors, which is done by several communities (e.g., LibreOffice,[3] KDE,[4] and Mozilla[5]). However, community managers argue that labeling issues manually is difficult and time-consuming [12]. For that reason, Huang et al. [32] proposes an approach for labeling good first issues. While this approach indicates easy issues for new contributors, it is as limited in the outcome as the approaches that only classify issues as bugs. In contrast, by labeling issues with domains of the APIs, our approach can support new contributors of different skill levels.

3 Method Overview

This section presents an overview of how we answered the research questions.

RQ.1: *How relevant are the API-domain labels to new contributors?* In this RQ (Section 4), we evaluate the manually curated labels with potential

---

[1] http://bit.ly/NewToOSS

[2] In this study, the words "tasks" and "issues" are used interchangeably.

[3] https://wiki.documentfoundation.org/Development/EasyHacks

[4] https://community.kde.org/KDE/Junior_Jobs

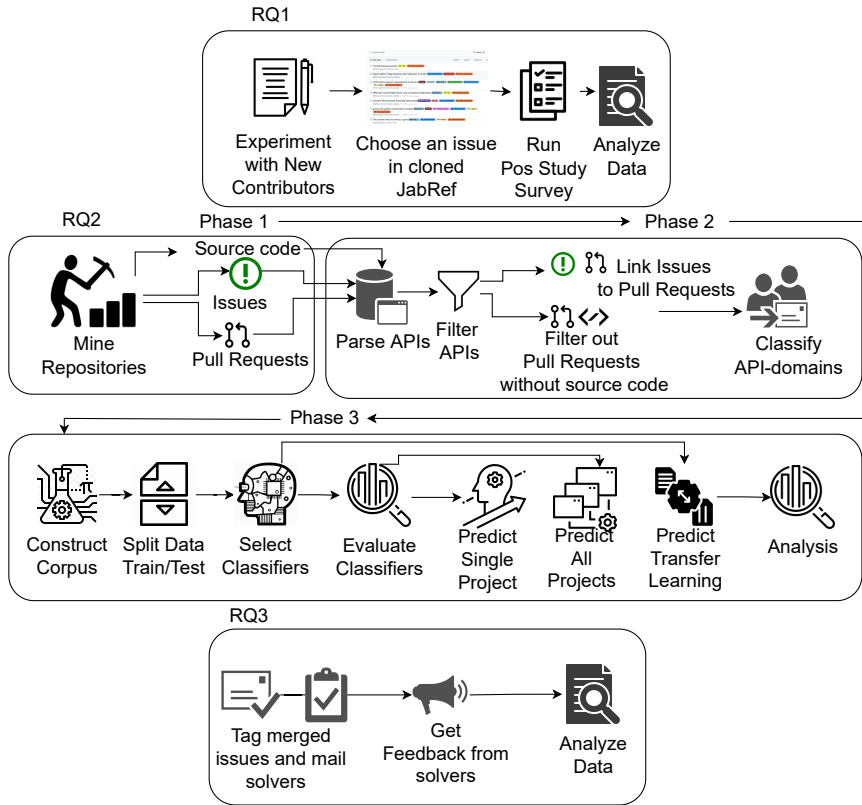[5] https://wiki.mozilla.org/Good_first_bug

Fig. 1 Research method overview

new contributors. We divided the participants into two groups. After mimicking the project's issues pages for 22 issues, we added API-domain labels to the issues for the treatment group and kept the page as-is for the control group. We asked the participants to select issues to which to contribute and fill out a survey about their selection process (Figure 1 - RQ1).

RQ.2: *To what extent can we automatically attribute API-domain labels to issues?* In this RQ (Section 5), we investigate the feasibility of predicting API-domain labels. We mined software repositories to collect issues, their associated pull requests, and the APIs used in the source code. Subsequently, we manually classified the APIs into API domains to build machine learning classifiers (Figure 1 - RQ2). To answer the sub-questions, we predicted the API-domain labels using each project dataset separately (RQ.2.1), a dataset with all projects merged (RQ.2.2), and different source and target datasets (RQ.2.3).

RQ.3: *How well do the API-domain labels match the skills needed to solve the issue?* Finally, In this RQ (Section 6), we asked contributors to provide

feedback on the usefulness of the labels that we predicted in identifying skills needed to complete the issue (Figure 1 - RQ3).

To foster reproducibility, we provide publicly available supplementary material[6] containing the raw data, the Jupyter notebook scripts, and the anonymized survey data.

4 Relevance of the Labels to New Contributors (RQ1)

4.1 Method

To explore the relevancy of the API-domain labels from an outsider's perspective, we conducted an experiment with 74 participants. We selected the JabRef project, hosted in GitHub, as the subject of the experiment. Two authors of this paper have already contributed to and have in-depth knowledge of the project. Having this knowledge helped us interpret the feedback and results. We created two versions of the JabRef issues page (with and without API-domain labels) and divided our participants into two groups (between-subjects design). We asked participants to choose and rank three issues to which they would like to contribute and answer a follow-up questionnaire about what information supported their decision. The artifacts used in this phase are part of the replication package.

4.1.1 Participants

We used convenience sampling by recruiting participants from both industry and academia. We reached out to instructors and IT managers of our personal and professional networks and asked them to help in inviting participants. From industry, we recruited participants from one medium-sized IT startup hosted in Brazil and the IT department of a large and global company. We recruited students from multiple universities, including undergraduate and graduate computer science students from one university in the US and two others in Brazil as well as graduate data science students from a university in Brazil, since they are also potential contributors to the JabRef project. Table 1 presents the participants' demographics. We offered an Amazon Gift card (US$ 25.00) to incentivize participation.

We categorized the participants' development tenure into novice and experienced coders, splitting our sample in half—below and above the average "years as professional developer". We also segmented the participants between industry practitioners and students. Participants are identified by a sequential number (column "Participant").

The participants were randomly split into two groups: Control and Treatment. Out of the 120 participants that started the questionnaire, 74 (61.7%) finished all the steps; we only consider these participants in our analysis. We

---

[6] https://doi.org/10.5281/zenodo.6869246

Table 1 Demographics Subgroups for the Experiment's Participants

Population	Quantity	Percentage	Tenure	Quantity	Percentage
Industry	41	55.5 %	Expert	19	25.7 %
Student	33	44.5 %	Novice	55	74.3 %

ended up with 33 and 41 participants in the Control and Treatment groups, respectively.

4.1.2 Experiment Planning

We selected 22 existing JabRef issues and built mock GitHub pages for Control and Treatment groups. The issues were selected from the most recent ones, trying to maintain similar distributions of the number of API-domain labels predicted per issue and the counts of predicted API-domain labels. The control group mockup page had only the original labels from the JabRef issues, and the treatment group mockup page presented the original labels in addition to API-domain labels. These pages are available in the replication package. We used a preliminary version of our prediction model to generate the API-domain labels [14].

4.1.3 Questionnaire Data Collection

The questionnaire included the following questions/instructions:

- Select the three issues that you would like to work on.
- Select the information (region) from the issue page that helped you decide which issues to select (Fig: 2).
- Why is the information you selected relevant? (open-ended question)
- Select the labels you considered relevant for choosing the three issues.
- What kind of label would you like to see in the issues? (open-ended question)

The questionnaire also asked about participants' experience level, experience as an OSS contributor, and expertise level in the technologies used in JabRef.

Figure 2 shows an example of an issue details page and an issue entry on an issue list page. After selecting the issues to contribute, the participant was presented with this page to select what information region was relevant to their issue selection.

4.1.4 Questionnaire Data Analysis

We split the analysis into two sets of questions.

Regions and Labels Choices Analysis. We first compared treatment and control groups' results to understand participants' perceptions about what

Fig. 2 Questionnaire question about the relevance of the page regions for task selection

information regions they considered important and the relevancy of the API-domain labels. We used violin plots to visually compare the distributions and measured the effect size using the Cliff's Delta test.

Then, we analyzed the data, aggregating participants according to their demographic information and resulting in the subgroups presented in Table 1. We calculated the odds ratio to check how likely it would be to receive similar responses from both groups. We used a 2x2 contingency table for each comparison—for instance, industry practitioners vs. students and experienced vs. novice coders. We used the following formula to calculate the odds ratio [33]:

$$OddsRatio(OR) = \frac{(a/c)}{(b/d)}$$

An odds ratio > 1 means that the first subgroup is more likely to report a type of label, while an odds ratio less than 1 means that the second group has greater chances (OR) [34].

Open Questions Analysis. To understand the rationale behind the label choices, we qualitatively analyzed the answers to the open questions ("Why was the information you selected relevant?" and "What kind of label would you like to see in the issues?"). We selected representative quotes to illustrate the participants' perceptions of the labels' relevancy.

We qualitatively analyzed the answers by inductively applying open coding in groups, where we identified the participant's reason for considering the provided information as relevant and what information the participant would like to be provided. We built post-formed codes as the analysis progressed and associated them with respective parts of the transcribed text to code the information relevance according to the participants' perspectives.

Researchers met weekly to discuss the coding. We discussed the codes and categorization until reaching a consensus about the meaning of and relationships among the codes. The outcome was a set of high-level categories as cataloged in our codebook[7].

4.2 Results

Information used when selecting a task. Understanding the type of information that participants used in their decision to select an issue can help projects better organize such information on their issue pages. Figure 3 shows the different regions that participants found useful. In the control group, the top two regions of interest included the title of the issue (78.8%) and the body (75.8%), followed by the labels (54.5%). This suggests that the labels used by the project were only marginally useful, and participants had to review the issue details. In contrast, in the treatment group, the top three regions of interest by priority were: title, label, and body 97.6%, 82.9%, 70.7%, respectively). This shows that participants in the Treatment group found the labels more useful than those in the control group: 82.9% usage in the treatment group compared to 54.5% in the control group. Comparing the body and the label regions in both groups, we found that participants from the treatment group selected 1.6x more label regions than the control group ($p<0.05$).

Our qualitative analysis reveals that the labels help in selecting issues. For instance, P2 mentioned: *"labels were useful to know the problem area and after reading the title of the issues, it was the first thing taken into consideration, even before opening to check the details"*. Participants found the labels to be useful in identifying the specific topic of the issue, as P4 stated: *"[labels are] hints about what areas have a connection with the problem occurring"*.

---

[7] https://doi.org/10.5281/zenodo.6869246
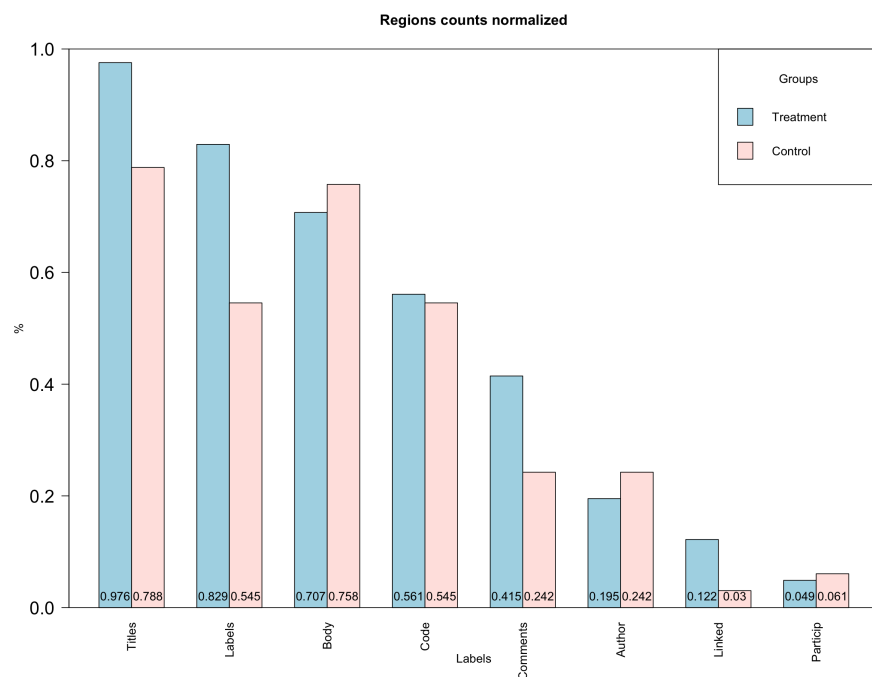
Fig. 3 The region counts (normalized) of the issue's information page selected as most relevant by participants from treatment and control groups.

Role of the labels. We also investigated which type of labels helped the participants in their decision-making. We divided the labels available to our participants into three groups based on the type of information.

– Issue type (already existing in the project): This included information about the type of the task: bug, enhancement, feature, good first issue, and GSoC (Google Summer of Code).
– Code component (already existing in the project): This included information about the specific code components of JabRef: entry, groups, external.files, main table, fetcher, entry.editor, preferences, import, keywords
– API-domain (new labels): the labels generated by our classifier (IO, UI, network, security, etc.). These labels were available only to the treatment group.

Table 2 Label distributions among the control and treatment groups

Type of Label	Control	Percentage	Treatment	Percentage
Issue Type	145	56.4 %	168	36.8 %
Components	112	43.6 %	94	20.6 %
API Domain	-	-	195	42.7 %

Table 3 Answers from different demographic subgroups regarding the API labels (API/-Component/Issue Type)

Subgroup	Comparison	API %	Comp or Type %
Industry	API/Comp	**56.0**	44.0
Students	API/Comp	40.0	**60.0**
Exp. Coders	API/Comp	**50.9**	49.1
Novice Coders	API/Comp	41.5	**58.5**
Industry	API/issue Type	45.5	**55.5**
Students	API/issue Type	30.6	**69.4**
Exp. Coders	API/issue Type	43.5	**56.5**
Novice Coders	API/issue Type	30.9	**69.1**

Table 2 compares the labels that participants considered relevant (Section 4.1.3) across the treatment and control groups distributed across these label types. In the control group, the most selected labels (56.4%) relate to the type of issue (e.g., Bug or Enhancement). In the treatment group, however, this number drops to 36.8%, with API-domain labels as the majority (42.7%), followed by code component labels (20.6%). This difference in distributions alludes to the usefulness of the API-domain labels.

To better understand the usefulness of the API-domain labels as compared to the other types of labels, we further investigated the label choices among the treatment group participants. Figure 4 presents two violin plots comparing (a) API-domain labels against code component labels and (b) API-domain labels against the type of issue. Wider sections of the violin plot represent a higher probability of observations taking a given value, the thinner sections correspond to a lower probability. The plots show that API-domain labels are more frequently chosen (median is 5 labels) as compared to code component labels (median is 2 labels), with a large effect size ($|d| = 0.52$). However, the distribution of the issue type and API-domain labels are similar as confirmed by negligible effect size ($|d| = 0.1$). These results indicate that while the type of issue (bug fix, enhancement, suitable for a newcomer) is important, understanding the technical (API) requirements of solving the task is equally important for developers deciding which task to select.

Finally, we analyzed whether the demographic subgroups held different perceptions about the API-domain labels (Table 3). When comparing industry vs. students, we found participants from industry selected 1.9x (p-value=0.001) more API-domain labels than students when we controlled by component labels. We found the same odds when we controlled by issue type (p-value=0.0007). When we compared experienced vs. novice coders, we did not find statistical significance (p=0.11) when controlling by component labels. However, we found that experienced coders selected 1.7x more API-domain labels than novice coders (p-value=0.01) when we controlled by the type of the issue.

The odds ratio analysis suggests that API-domain labels are more likely to be perceived as relevant by practitioners and experienced developers than by students and novice coders.
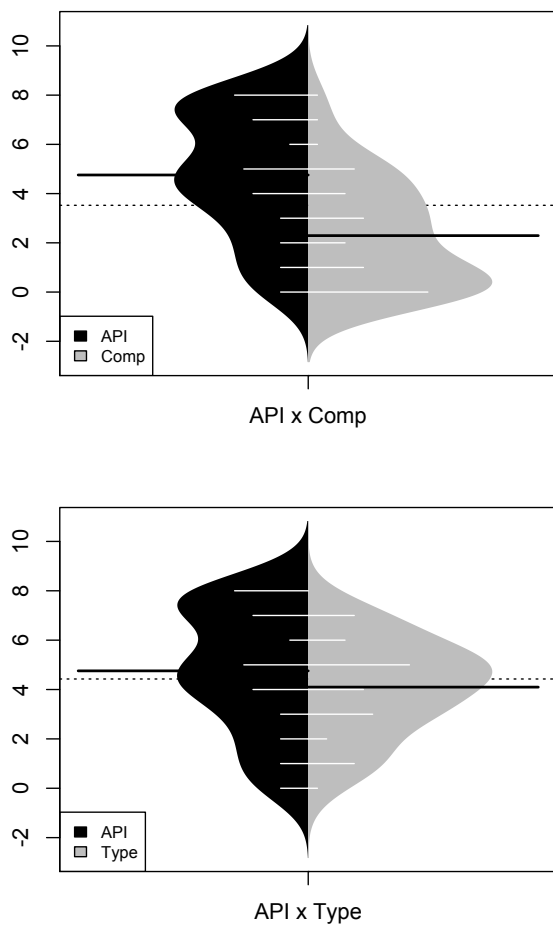
Fig. 4 The Y-Axis contains the density probability and the median of API-domain labels (API) x Component labels (Comp) x Type labels

The way contributors analyzed the issues. We used the questionnaire's open-ended question to evaluate how subjects used the information to decide whether the task was appropriate to them (Section 4.1.4).

Our qualitative analysis revealed a set of 22 categories of information reported as relevant by contributors when they decide on a task to which to contribute. We organized the 22 categories of information based on an existing model from literature, the 5W2H framework, as we explain below and illustrate in Figure 5. The 5W2H framework (5-Wh and 2-How questions) is often used for clarifying a problem, issue, error, or nonconformity, or to facilitate implementing effective actions. The framework was initially applied to

the automotive and other manufacturing industries [35] and later to quality management [36] and software engineering [37].
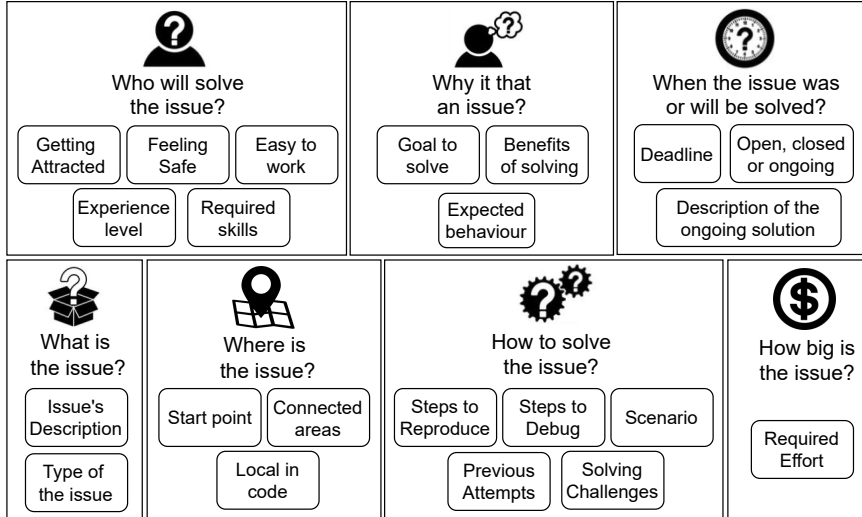
Fig. 5 The information reported by contributors as relevant to choosing a task. We mapped the categories of our participants' definitions (rounded squares) to the 5W2H framework [37], which organizes information for decision-making across seven questions.

___Who will solve the issue?___ This category contains information about the forces influencing people to choose to work on an issue. Contributors mentioned what can influence one's decision to select the issue. A newcomer can BECOME ATTRACTED to select the issue when *"filtering labels to search issues that [they] would like to contribute the most"* (P34) and reading the title (P18) to see if it *"includes something that is not too wordy and if it uses words [they] could easily understand"* (P21). When opening the issue, participants also reported the body and the comments were relevant to *"gain interest on the issue"* (P4) , as a *"detailed body and helpful comments from experienced people in the project is extremely helpful to make the newcomer* FEELING SAFE *to try the issue"* (P6). The contributors' confidence to decide about an issue can increase when they match their EXPERIENCE LEVEL with the indication of difficulty to solve the issue (P8, P4) which could be shown in a label such as "easy, medium, hard" (P4) , "good first issue" (P6), or "good challenging issue" (P7). Besides their experience, contributors can use the REQUIRED SKILLS to work on the issue to judge *"if they have the [necessary] skill to help"* (P31) or *"whether or not [are] capable of finding a solution"* (P32). The required technical skills mentioned by participants included the programming language of the code (P21, P27, P33), the architecture layer - front-end, back-end, interface (P27, P3, P2), APIs (P41, P42), database (P33), frameworks, and libraries (P20).

<u>Why is that an issue?</u> is a category that justifies the issue as an issue. Participants mentioned the reasoning for an issue to exist could raise interest in new contributors, such as knowing the GOAL TO SOLVE and *"what is the purpose of the issue"* (P44), and the BENEFITS OF SOLVING or *"why solving it will help users"* (P45). Additionally, the EXPECTED BEHAVIOR of the software can help to clarify why the reported issue is an issue in comparison to the normal behavior of the software. Hence, the expected behavior represents a *"critical information to decide what is happening in the system and what is expected"* (P61). Indeed, one participant reported: *"I would only contribute something that I know how it works"* (P22).

<u>When was the issue solved, or when will it be solved?</u> introduces time-related information and constraints regarding the issue. Participants reported they would like to know the DEADLINE TO SOLVE the issue or the *"urgency"* (P13). Participants suggested that the priority appear in a label (P17) and be defined according to the impact that the issue has on businesses or users (P15). Another issue related to time is the *"status to check the issue's state"* (P33), which can be OPEN, CLOSED, OR ONGOING, allowing contributors to use a filter in the issues' page. Since they "don't look at closed issues much, [...] the open flag grabs [their] attention" (P43). When a contributor is currently working on a solution, they should have their names assigned to the issue and include a comment with the DESCRIPTION OF AN ONGOING SOLUTION that should "demonstrate the issue's status" (P35).

<u>What is the issue?</u> relates to the description of the issue itself. Participants raised the importance of clear ISSUES' DESCRIPTION, including both a summarized *"idea of what the issue is about"* (P28) and a comprehensive explanation *"to help understand what is the problem"* (P45) about. When an issue provides both levels of details, it *"tells about the problem, first in a general term and later giving [them] details about it"* (P12). The issue's TYPE in labels *"demonstrate [...] how [the issue] is classified"* (P35). The participants suggested the issue should have *"labels that inform precisely which type of issue is"* (P40): bug (P41), a new feature (P42), performance (P42), enhancement (P42), and security. One participant (P43) emphasized that *"'all issues should have a type so [they] can see if [their] skill set is useful"* (P43).

<u>Where is the issue?</u> references the localization of the issue in the code or project, guiding contributors to a START POINT or *"where to start looking at in the code/library to investigate the problem"* (P4). The LOCAL IN CODE or the code block, method, or class which is causing the issue, and CONNECTED AREAS. This information would *"give some hints about what areas have a connection with the problem occurring"* (P4) and *"code snippet to provide context for wherein the program this issue was happening"* (P18).

<u>How to solve the issue?</u> brings practical directions to guide solving the issue. Awareness of *"PREVIOUS ATTEMPTS to solve [an issue]"* (P30) helps contributors with *"valuable information about what has already been done and properly documented"* (P42). Contributors who are deciding about an issue can read *"[SOLVING] CHALLENGES"* (P35) to avoid wasting time on previous

attempts and focus their effort on new paths to achieve the solution. When working on the issue, having STEPS TO REPRODUCE the error (P45) on a controlled environment also help to solve the issue. Participants also mentioned they would like to see *"linked issues and comments to help understand the* SCENARIO*"* (P33), and STEPS TO DEBUG to *"decipher what the problems really is"* (P41).

How big is the issue? is information that can provide visibility of the REQUIRED EFFORT for *"[a contributor] to work on alone until [they] solve it"* (P7). If the issue does not have this information, the developer tries to *"grasp what's the idea of the issue, to better measure how long it would take to solve it"* (P19).

Finally, the question "What region has this information?" identifies the regions where the participants found the information in this study. Title appeared 7 times, body 8, comments 13, labels 5, status 2, code snippet 3, and linked issue 2.

5W2H outcomes: the analysis confirmed the relevance of the title, body, comments, and labels and helped to create a taxonomy of what contributors analyze when deciding whether they want to contribute to an issue. The qualitative code we built for this open-ended question may be explored in future work to create ways to show the contributor such information using templates, labels, bots, or other UI objects.

Preferred types of labels. Towards the evaluation of the labels contributors want to see in issues pages, 42 participants (out of 74) answered the open question Q2 ("What kind of label do you want to see in the issues?"). The TYPE, PRIORITY to solve, and PROGRAMMING LANGUAGE *"in which the code was written in"* (P21) were the three most mentioned, followed by DIFFICULTY LEVEL, TECHNOLOGY, and API. Some participants suggested different semantics for the label TYPE: bug (P3, P41), improvement (P3), performance (P35, P42), new feature (P42), or security (P36). Other participants also suggested different semantics for DIFFICULTY LEVEL: *"good first issue"* (P6), *"good challenging issue"* (P7), or *"easy, medium, hard"* (P4). The semantics for each label can be explored in future work. We present the 11 categories of suggested labels that we qualitatively coded from the participants' answers in Table 4.

Participants prefer to see labels on priority, type, programming language, complexity, technology, and APIs more than architecture, status, GitHub info, database, and framework. GitHub info is general information about the project repository (e.g., "ranking about the most commented" (P10P0) and "branches" (P22I0)).

Table 4 Labels desired by participants to select the issue

Group	Desired Labels	Participants who Mentioned
Management	Type	P41, P35, P42, P3, P36, P37, P38, P39, P43, P40
Management	Priority	P12, P13, P19, P20, P14, P15, P16, P29, P17, P18
Technical	Programming language	P34, P33, P21, P22, P27, P23, P24, P22, P26
Management	Difficulty level	P4, P5, P6, P7, P9, P8
Technical	Technology	P30, P34, P33, P32, P20, P31
Technical	API	P41, P42, P3, P1, P20
Technical	Architecture layer	P2, P3, P27, P18
Management	Status	P28, P9, P29
Management	GitHub info	P10, P19
Technical	Database	P33
Technical	Framework	P20

> **RQ.1 Summary.** Our findings suggest that labels are relevant for select-
> ing an issue to work on. API-domain labels increased the perception of
> the labels' relevancy. API-domain labels are especially relevant for indus-
> try and experienced coders. API is one of the issue labels users want to
> see. 5W2H analysis has confirmed the relevance of labels and can guide
> contributors on how to write an issue.

5 Label Predictions (RQ2)

Even with the relevance of the API-domain labels, we investigated how to
predict them automatically.

5.1 Method

To predict the API-domain labels, we employed a multi-label classification
approach. This approach is divided into three phases: phase 1 - mining the
repositories; phase 2 - parsing the source code and semi-automatically catego-
rizing the APIs with experts; and phase 3 - building the corpus and running the
classifiers (Figure 1). Additionally, we explored the influence of issue elements
(i.e., title, body, and comments) and machine learning setup (i.e., n-grams and
different algorithms) on the predictions.

   In our preliminary work [14], we conducted an exploratory experiment on a
single project (JabRef). In the current study, we include four new projects. We
selected projects to increase the diversity of domains, programming languages,
and human languages (vocabularies). We sought a mix of popular open-source
(OSS) and closed-source currently active projects with a large number of issues
and pull requests. As we aimed to run surveys within the project communities,

Table 5 Project Details. R - Number of Releases. C - Number of Contributors

Project	R	C	Stars	Forks	Closed Pulls	Issues	Domain	OSS
JabRef	42	337	15.7K	1.8K	2.7K	4.1K	Articles manager	Y
Audacity	25	154	6.9K	17.1K	1.1K	0.6K	Audio editor	Y
PowerToys	50	262	65.5K	3.7K	3.3K	9.9K	Utilities for Windows	Y
RTTS	121	40	N.A.	N.A.	N.A.	N.A.	Telecommu- nication product	N
Cronos	123	–	N.A.	N.A.	N.A.	N.A.	Time Tracker	N

we contacted maintainers/managers of candidate projects in advance to explain our goals and seek support in reaching contributors for the user studies. Table 5 presents the selected projects and their characteristics.

The study can be divided into two branches of prediction: TF-IDF and BERT. The TF-IDF predictions followed the previous study [14], employing five algorithms (Random Forest Decision Tree Logistic Regression, MLP Classifier, and MLkNN) but were extended to more projects, ITSs, programming languages, and vocabularies (natural languages). The BERT predictions operate the same extensions but are restricted to English vocabulary. Unlike the TF-IDF, BERT determines the meanings of words in a corpus based on their context within a sentence. We compared BERT to the previous TF-IDF classification pipeline within the context of the issue labeling problem.

5.1.1 Phase 1 - Mining Software Repositories

We started by gathering data from the repositories to train a machine learning model to predict the API labels. To achieve this goal, we mined closed issues and merged pull requests. Table 6 summarizes the projects' characteristics and demographics. We collected a total of 22,231 issues and 4,674 pull requests (PR) from all projects, considering all project data until November 2021. For the OSS projects, we used the GitHub REST API v3 to collect data such as title, body, comments, and closure date. We also collected the name of the files changed in the PR and the commit message associated with each commit. The industry projects used Gerrit (RTTS - Real-Time Telecom Software) and Jira + MTT - Minds At Work Time Tracker (Cronos). From RTTS, we extracted two CSVs files: one containing the "issues" (troubles in RTTS) and the second containing the commits. The Cronos project uses a combination of Jira to track the open issues and the software MTT, an in-house solution, to manage the revisions and allocation time. We extracted a CSV file from Jira and a TXT from MTT.

Next, to train the model, we kept only the data from issues linked with merged and closed pull requests, since we needed to map issue data to source

Table 6 Projects Mined and Issue Tracker Systems

Project	Prog Lang	Issue Tracker/ Vocab- ulary	Extraction Method	Issues/ PR	Linked Issues & PR	Source Code Files	Distinct APIs
JabRef	Java	GitHub EN	GitHub API V3	4,471 1,966	1,914	1,690	1,944
Audacity	C++	GitHub EN	GitHub API V3	1,440 310	341	624	1,478
PowerToys	C#	GitHub EN	GitHub API V3	12,571 853	1,011	794	264
RTTS	Java	Gerrit EN	Export CSV	2,836 470	470	9,779	8,645
Cronos	Java	Jira/MTT BR	Export CSV/TXT	913 1075	206	220	441
Total				22,231 4,674	3,942	13,107	12,772

code APIs. To find the links between pull requests and issues in open source projects, we searched for the symbol #issue_number in the pull request title and body and checked the URL associated with each link. We also filtered out issues linked to pull requests without at least one source code file (e.g., those associated only with documentation files) since they do not provide the model with content related to any API. Similarly, we linked projects hosted by Gerrit and Jira/MTT, using the trouble ID and key fields (Gerrit), and for the project managed with Jira/MTT, we linked using the change ID and revision fields. The TXT file from MTT needed to be parsed to look for the revision field. We discarded entries without source code or linked data. In total, 734 entries were discarded.

5.1.2 Phase 2 - API classification

Phase 2 encompasses API extraction and expert classification.

API extraction. To identify the APIs used in the source code affected by each pull request, we built a parser to process all source files from the projects. In total, 12,772 library declaration statements from 13,107 source files were mapped to 185,159 possible relationships between files and APIs. The parser looked for specific commands, i.e., import (Java), using (C#), and include (C++). The parser identified all classes, including the complete namespace from each import/using/include statement. We considered only the most frequent language per project.

Then, we filtered out APIs not found in the latest version of the source code (JabRef 5.3, Audacity 3.1.0, and PowerToys 0.49.1; RTTS and Cronos are industry projects, and we used the last provided version) to avoid recommending APIs in source code that were no longer used in the project. The filtering process is automatic. When processing a closed pull request, the files attached have their filenames compared with those stored in a database by the parser. When the file name is not found in the database, the pull request is discarded from the training set.

Our final dataset comprises 22,231 issues, 4,674 pull requests, 13,107 files, and 12,772 distinct APIs (Table 6).

Table 7 Labels Definition

Label Generated	Definition
Application (App)	Third-party apps or plugins for specific use attached to the System
Application Performance Manager (APM)	Monitors performance or benchmark
Big Data	APIs that deal with storing large amount of data, with variety of formats
Cloud	APIs for software and services that run on the Internet
Computer Graphics (CG)	Manipulating visual content
Data Structure	Data structures patterns (e.g., collections, lists, trees)
Databases (DB)	Databases or metadata
Software Development and IT Operations (DevOps)	Libraries for version control, continuous integration and continuous delivery
Error Handling	Response and recovery procedures from error conditions
Event Handling	Answers to events like listeners
Geographic Information System (GIS)	Geographically referenced information
Input-Output (IO)	Read, write data
Interpreter	Compiler or interpreter features
Internationalization (i18n)	Integrate and infuse international, intercultural, and global dimensions
Logic	Frameworks, Patterns like Commands, Controls or architecture-oriented classes
Language (Lang)	Internal language features and conversions
Logging	Log registry for the app
Machine Learning (ML)	ML support like build a model based on training data
Microservices/Services	Independently deployable smaller services. Interface between two different applications so that they can communicate with each other
Multimedia	Representation of information with text, audio, video
Multi-Thread (Thread)	Support for concurrent execution
Natural Language Processing (NLP)	Process and analyze natural language data.
Network	Web protocols, sockets, RMI APIs
Operating System (OS)	APIs to access and manage a computer's resources
Parser	Breaks down data into recognized pieces for further analysis.
Search	API for web searching
Security	Crypto and secure protocols
Setup	Internal app configurations
User Interface (UI)	Defines forms, screens, visual controls
Utility (Util)	Third-party libraries for general use
Test	Test automation

Expert Classification. Three software engineering experts (senior developers), including one of the authors of this article, proposed the labels based on their experience in software development, considering possible categories generic enough to suit a wide range of APIs present in software projects. For

example, the proposed domains contain UI, IO, Cloud, Error handling, etc. After four rounds of discussions, the experts reached a consensus, and 31 API domains were defined (Table 7).
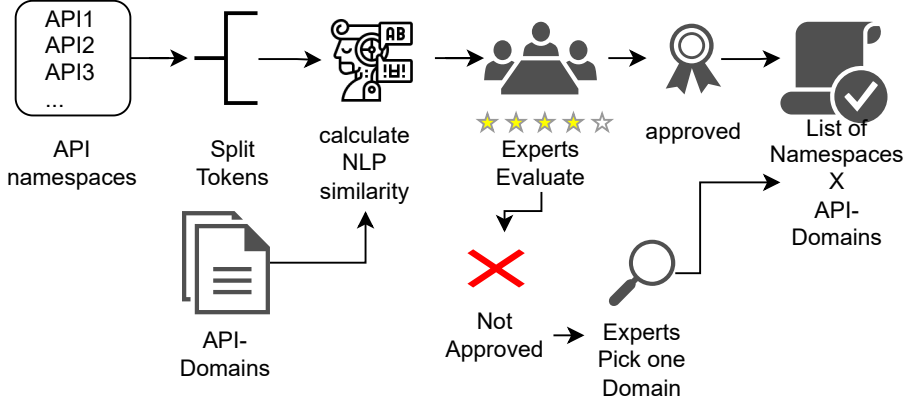
Fig. 6 Process for evaluating APIs by experts

After defining the 31 API domains, we started to classify the APIs semi-automatically (Figure 6). The intuition behind the API classification method is that libraries' namespaces often reveal architectural information and, consequently, their categories or API domains [38, 39]. To identify the possible API domains for each API, we split all the API namespaces into tokens. For instance the API "com.oracle.xml.util.XMLUtil" was split in "com", "oracle", "xml", "util", and "XMLUtil". Next, we eliminated the business domain name extensions (e.g., "org", "com"), country code top-level domain ("au", "uk", etc.), and the project and company names ("microsoft", "google", "facebook", etc.). In the example, we kept the first token "xml", second token "util", and full namespace "com.oracle.xml.util.XMLUtil."

For each token, we identified how similar it is to the 31 proposed API domains using an NLP similarity function. The intention is to suggest to the experts potential fits for the APIs. We used the NLP Python package spacy [40]. Spacy is a multi-use NLP package and can retrieve the semantic similarity of words using word2vec. We set up the spacy package with the largest trained model available (large full vector package, en_core_web_lg, which includes 685k unique vectors).

To assist the expert evaluation and reduce the search scope, we aggregated the tokens found in namespaces. For instance, to evaluate the APIs for the Cronos project, the experts received a list with 32 "first tokens" and a list with 73 "second tokens" automatically aggregated using SQL commands for each token. Finally, the experts analyzed the complete list (tokens + similarity suggestions) to pick one suggestion or decide using their experience. The whole process is illustrated in Figure 6 and exemplified below. Table 8 shows the number of APIs evaluated by the experts in two rounds after the aggregations.

Therefore, instead of classifying the 441 APIs found in Cronos source code, they checked the NLP suggestions in the list of first and second tokens.

Table 8 Number of APIs classified per project

Project	Total APIs	APIs submited to experts 1st token	APIs submited to experts 2nd token	APIs % 1st round	APIs % 1st + 2nd round
Audacity	1,478	562	106	38.0%	45.1%
PowerToys	264	37	20	14.0%	21.6%
Rmca	8,645	10	95	1.7%	2.3%
Cronos	441	32	73	7.2%	23.8%
JabRef	1,692	137	45	8.0%	10.8%
Total / avg	12,520	869	339	6.9%	9.64%

The process employed three experts (one author and two senior developers) and a card-sorting approach to manually accept or reject the suggestions for each token in the list. Each expert picked up one of the suggestions or chose a better API domain based on their experience. The experts could also check the list of full namespaces if they did not agree with the NLP suggestions. For example, considering the namespace "com.oracle.xml.util.XMLUtil:" for the first token, "xml", the similarity function suggested possible API-domain labels and a similarity value: Input and Output: 0.7, Error Handling: 0.69, Parser: 0.57. For the token "util", it suggested: Utility: 0.9, Data Structure: 0.49. Therefore, the namespace "com.oracle.xml.util.XMLUtil" was classified as "Utility." The majority of the APIs were classified using the first or second token. In a few cases ($< 10\%$), the experts had to classify the full namespace. After classifying all the tokens, the experts conducted a second round to achieve consensus ($\sim$16 hours for all projects).

The project in Portuguese followed the same expert classification process employed in English projects. Indeed, the libraries declared in the Cronos project source code are written using English words and therefore did not harm the NLP categorization.

We used these 31 categories (API-domains labels) for the 22,231 issues previously collected based on the presence of the corresponding APIs in the changed files. We used this annotated set to build our training and test sets for the multi-label classification models.

5.1.3 Phase 3 - Building the Multi-label Classifiers

Since solving an issue may require multiple types of APIs, we applied a multi-label classification approach, which has been used in software engineering for purposes such as classifying questions in Stack Overflow (e.g., [18]) and detecting types of failures (e.g., [41]) and code smells (e.g., [42]). To build the classifiers, we first needed to build the corpus and then run and evaluate the classifiers.

Corpus construction. The corpus construction comprised pre-processing, cleaning, diagnostics, and splitting into training and test datasets.

Pre-processing: We built two distinct models—one that uses TF-IDF [43] and another that uses BERT [44]. These corpora include the issue title, body, and comment texts of the selected issues.

Next, similar to other studies [43, 45, 46], we applied TF-IDF, which is a technique for quantifying word importance in documents by assigning a weight to each word. After applying TF-IDF, we obtained a vector of TF-IDF scores for each issue's word. The vector length is the number of terms used to calculate the TF-IDF, and each term received the TF-IDF score. These TF-IDF scores are then passed to one of the selected classifiers (e.g., RandomForest) to label each issue. Each label receives a binary value (0 or 1), indicating whether the corresponding API domain is present in the issue.

For BERT, we created two separate CSV files: an input binary with expert API-domain labels paired with the issue corpus, as well as a list of the possible labels for the specific project. BERT directly labels the issue with the corpus text and labels list without the need for an additional classifier.

We also evaluated the classifier's performance by combining in one dataset all the projects that use English vocabulary. Therefore, we also had to build a new composed ID (ID + project name) for all projects to guarantee uniqueness. For this experiment, after we created the new IDs, we merged the binaries of the project, including the classes missing for each project (RTTS does not have a Computer Graphics label, for example). We compared various algorithms to identify the best setup.

Cleaning: To build our classification models using TF-IDF, we converted each word in the corpus to lowercase and removed URLs, source code, numbers, and punctuation. We also removed stop-words and stemmed the words using the Python nltk package. We filtered out the issue and pull request templates[8] since their repetitive structure introduced noise and were not consistently used among the issues.

We follow the work of Izadi et al. [47] to process data for BERT. We tested BERT with a cleaned and uncleaned corpus. This was checked by comparing the F-measure, precision, and recall results from training with cleaned and uncleaned corpora. We ran three training trials with a 10-fold ShuffleSplit CV and determined that the unclean corpus consistently delivered higher metrics than any cleaning method (stemming, removing stopwords, etc.) The result is in line with Izadi et al. [47] who showed that an unclean input corpus best maintained the context of words needed for BERT to determine their meaning and significance.

Diagnostics: Multi-label datasets are usually described by label cardinality and label density [48]. Label cardinality is the average number of labels per sample. Label density is the number of labels per sample divided by the total number of labels, averaged over the samples. For our dataset, the label cardinality is 8.19 and the density is 0.26. These values consider the 22,231

---

[8] `http://bit.ly/NewToOSS`

distinct issues and API-domain labels obtained after the previous section's pre-processing steps. Since our density can be considered high, the multi-label learning process or inference ability is not compromised [49].

Training/Test Sets: We split the data into training and test sets using the ShuffleSplit method [48], which is a model selection technique that emulates cross-validation for multi-label classifiers. For example, in the JabRef project, we had 1,914 linked issues, and since one PR could be linked with more than one issue, we kept 1,648 entries that we randomly split into a training set with 80% (1,318), 70% (1,154), and 60% (989) of the issues and a test set with the remaining 20% (330 issues), 30% (494), and 40% (659). We ran each experiment ten times, using ten different training and test sets to match 10-fold cross-validation. To improve the balance of the data set, we ran the SMOTE algorithm for the multi-label approach [50].

Classifiers. To create the classification models, we chose six classifiers that work with the multi-label approach and implemented different strategies to create learning models: Decision Tree, Random Forest (ensemble classifier), MLPC Classifier (neural network multilayer perceptron), MLkNN (multi-label lazy learning approach based on the traditional K-nearest neighbor algorithm) [51, 48], Logistic Regression, and BERT. We ran the first five classifiers using the Python sklearn package and tested several parameters. For the RandomForestClassifier, the best classifier, we kept the following parameters: *criterion = 'entropy', max_depth = 50, min_samples_leaf = 1, min_samples_split =3, n_estimators = 50.*

The BERT model was built using the open-source python package, Fast-Bert [52], which builds on the Transformers [53] library for Pytorch. Before training the model, the optimal learning rate was computed using a lamb optimizer [54]. Finally, the model fit over 11 epochs and validated every epoch. This training and validation occurred for every fold in the ShuffleSplit 10-fold cross-validation. The BERT model was trained on an NVIDIA Tesla V100 GPU that is contained within a computing cluster. The choice of hardware is not critical so long as the target GPU has sufficient VRAM to train the BERT model.

Classifiers Evaluation: To evaluate the classifiers, we employed the following metrics (also calculated using the scikit-learn package):

- **Hamming loss** measures the fraction of wrong labels to the total number of labels.
- **Precision** measures the proportion between the number of correctly predicted labels and the total number of predicted labels.
- **Recall** corresponds to the percentage of correctly predicted labels among all relevant labels.
- **F-measure** calculates the harmonic mean of precision and recall. F-measure is a weighted measure of how many relevant labels are predicted and how many of the predicted labels are relevant.

$$Precision = \frac{TP}{TP + FP} \qquad \text{(i)}$$

$$Recall = \frac{TP}{TP + FN} \qquad \text{(ii)}$$

$$FMeasure = \frac{2TP}{2TP + FP + FN} \qquad \text{(iii)}$$

The classic formulas to compute precision (i), recall (ii), and F-measure (iii) based on TP, TN, FP, and FN (true positives, true negatives, false positives, and false negatives) traditionally address single-label problems. An instance is considered correct or incorrect in single-label problems, while an instance may be partially correct in a multi-label evaluation; i.e., only a subset of the classes is correct for some instances. To address the multi-label classification problem, the literature [55] suggests adapting the aforementioned metrics as follows.

The metrics for each label can be calculated using different averaging strategies, as described in the following formulas. Let TP_l, FP_l, TN_l, and FN_l be the number of true positives, false positives, true negatives, and false negatives returned by a binary evaluation effort $B(TP, TN, FP, FN)$ such as the binary relevance transformation for a label l [56] and q is the number of labels. The macro averaging [55] is the arithmetic mean of all the per-label metrics, while micro averaging [55] is the global average metric obtained by summing TP, FN, and FP. The averages are computed and used to calculate the precision, recall, and F-measure (i, ii, iii). Santos et al. [14] used micro averaging to calculate the predictions' metrics. Thus, we kept it to compare with the previous study. The micro average favors the most populated classes [57].

$$B_{macro} = \frac{1}{q} \sum_{l=1}^{q} B(TP_l, FP_l, TN_l, FN_l) \qquad \text{(iv)}$$

$$B_{micro} = B\left(\sum_{l=1}^{q} TP_l, \sum_{l=1}^{q} FP_l, \sum_{l=1}^{q} TN_l, \sum_{l=1}^{q} FN_l \right) \qquad \text{(v)}$$

Transfer Learning. Next, we investigate the behavior of the metrics when we use different sets to train and test the model. We combined four projects using English vocabulary using three projects for training and one for testing. For instance, we trained a dataset with JabRef, PowerToys, and Audacity to test using the RTTS project. Next, we substituted the test dataset with one in the training set until completing all possible combinations.

Data Analysis. We used the aforementioned evaluation metrics, and the confusion matrix logged after each model's execution to evaluate the classifiers. We used the Mann-Whitney U test to compare the classifier metrics, followed by Cliff's delta effect size test. The Cliff's delta magnitude was assessed using the thresholds provided by Romano et al. [58], i.e. $|d|<0.147$ "negligible", $|d|<0.33$ "small", $|d|<0.474$ "medium", otherwise "large". We considered p-value < 0.05 as the limit to determine a statistical difference.

For the remainder of our analysis, we filtered out the API labels with no occurrence. "Cloud" and "Machine Learning" did not appear in any issues/PR mined and, therefore, had no predictions. We also filtered out labels that appeared in more than 90% of rows when running models for each project. Those could bias our predictions, since the classifier could always suggest them. For PowerToys, for example, the labels "NLP", "Network", "DB", "Error Handling", "Language", "DevOps", "IO", "ML", "Security", "Cloud", "Event Handling", "CG", "Multimedia", "Thread", "Big Data" and "GIS" had no occurrences and therefore were removed. The label "Util" was also removed because it surpassed the labels threshold (presented more than 90% of the rows in the dataset). The "Util" label was the most present with 699 occurrences, followed by the 501 occurrences of "App" and 498 of "UI". The less representative set had "Test" (6), "Logging" (6), and "i18n" (4). Power-Toys is a set of utility tools for Microsoft Windows. The high frequency of "Util" labels is expected.

The predictions using the dataset with all projects considerably changed our distribution of labels. The most frequent Labels were "UI" with 762 occurrences, followed by "Util" with 726 and "Logic" with 575. The less frequent labels were: "NLP" (45), "CG" (16), and "GIS" (10). Despite some labels being popular and having been used for tagging many APIs by the experts, the lack of pull requests submitted that touched source codes with those APIs may explain their rareness. The lack of linked issues and pull requests that mention those labels can also cause the absence in the dataset. Finally, training all the datasets together helped to spread the labels' frequency, for instance: "Util" and "Logic" labels were dropped when training the JabRef project because they reached the threshold of 90% of label predictions. When training using the dataset with all projects combined, those labels prevailed, staying below the 90% threshold, and were used to tag the issues (Figure 7).

Finally, we checked the distribution of the number of labels per issue (Figure 8). We found 110 issues with six labels, 106 issues with three labels, 104 issues with seven labels, and 102 issues with eight labels. Only 4.1% (=40) of issues have one label, which confirms a multi-label classification problem (Figure 8).
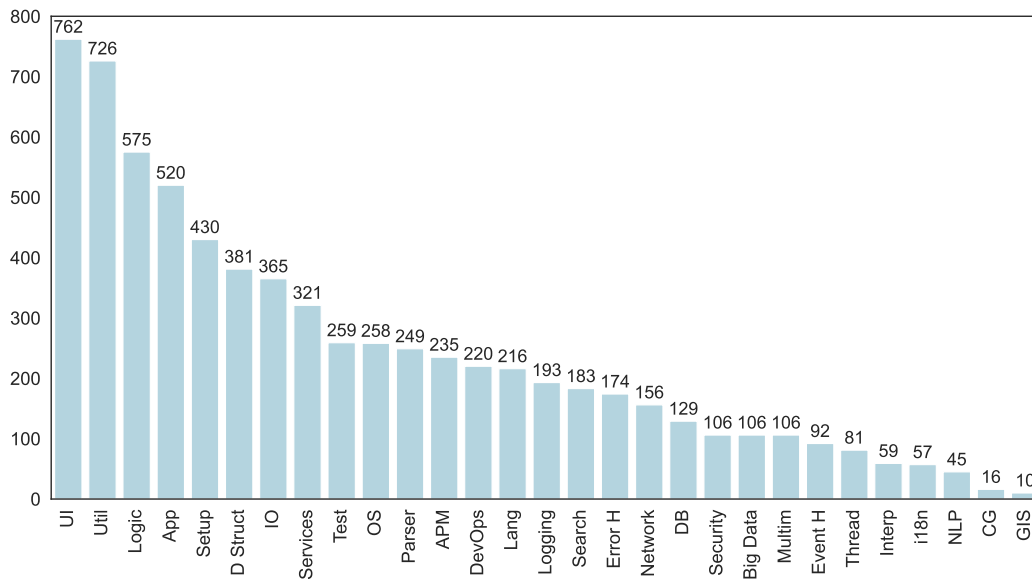
Fig. 7 Number of labels per type from the model with all the datasets merged
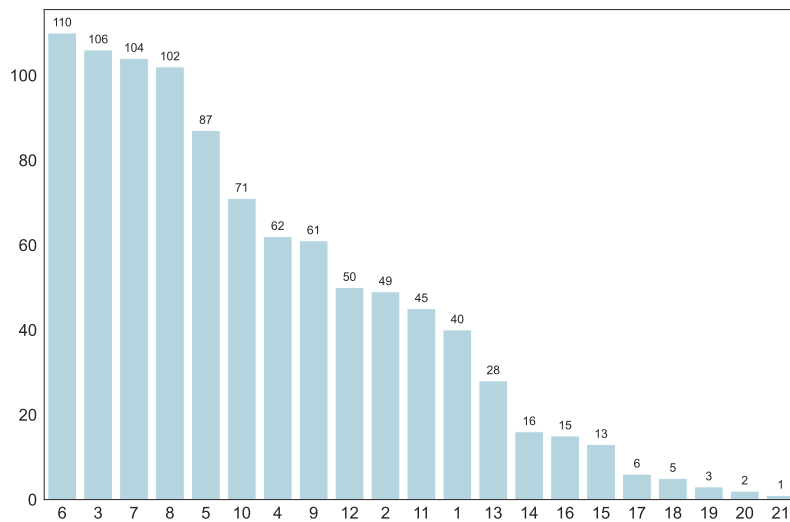

Fig. 8 Number of labels per issue from the model with all the datasets merged

5.2 Results

RQ.2.1: To what extent can we automatically attribute API-domain labels to issues using data from the project?

To predict the API-domains labels, we started by testing a simple corpus: only the issue TITLE as input and the Random Forest (RF) algorithm, since it is insensitive to parameter settings [59] and has shown to yield good prediction results in software engineering studies [60, 61, 62, 63]. Then, we evaluated the corpus configuration alternatives, varying the input information: only TITLE (T), only BODY (B), TITLE and BODY (T+B), and TITLE, BODY, and COMMENTS (T+B+C) comparing the average of all projects. To compare the different corpus configuration, we kept the Random Forest algorithm and used the Mann-Whitney U test with the Cliff's-delta effect size.

We also tested alternative configurations using n-grams. For each step, the best configuration was kept. Then, we used different machine learning algorithms and compared them to a dummy (random) classifier.

As Figure 9 and Table 17 (Appendix A) show, when we tested different inputs and compared them to TITLE only, all alternative settings provided better results with TF-IDF. We observed improvements in terms of precision, recall, and F-measure from the previous study [14]. When using BODY, we reached a precision of 84%, recall of 78.6%, and F-Measure of 81.1%. In contrast, while BERT had worse results, the model with the TITLE outperformed the other BERT models with 61.6% precision.

Table 9 Cliff's Delta for F-Measure and Precision: comparison of corpus model alternatives for TF-IDF and BERT. Title(T), Body(B) and Comments (C).

TF-IDF/BERT	Corpus Comparison	Cliff's delta			
		F-measure		Precision	
TF-IDF	T versus B	-0.005	negligible	-0.15	small***
TF-IDF	T versus T+B	-0.10	negligible***	-0.12	negligible***
TF-IDF	T versus T+B+C	-0.03	negligible***	-0.01	negligible
TF-IDF	B versus T+B	0.10	negligible***	0.02	negligible
TF-IDF	B versus T+B+C	-0.02	negligible	0.14	negligible***
TF-IDF	T+B versus T+B+C	0.07	negligible***	0.11	negligible***
BERT	T versus B	0.07	negligible	0.11	negligible
BERT	T versus T+B	0.13	negligible	0.03	negligible
BERT	T versus T+B+C	0.03	negligible	0.09	negligible
BERT	B versus T+B	0.10	negligible	-0.04	negligible
BERT	B versus T+B+C	-0.006	negligible	0.08	negligible
BERT	T+B versus T+B+C	-0.09	negligible	-0.01	negligible

*$p \leq 0.05$; ** $p \leq 0.01$; *** $p \leq 0.001$

For TF-IDF, we found statistical differences comparing the results using TITLE only and all the three other corpus configurations: F-measure (p-value ≤ 0.001 when comparing with TITLE+BODY or TITLE+BODY+COMMENTS, Mann-Whitney U test) and precision (p-value ≤ 0.001 when comparing with BODY or TITLE+BODY, Mann-Whitney U test), both with negligible effect size when comparing the precision from TITLE and BODY. The corpus configured with BODY performed better than all others in terms of precision, followed closer by the one set up with TITLE+BODY, which performed better in re-
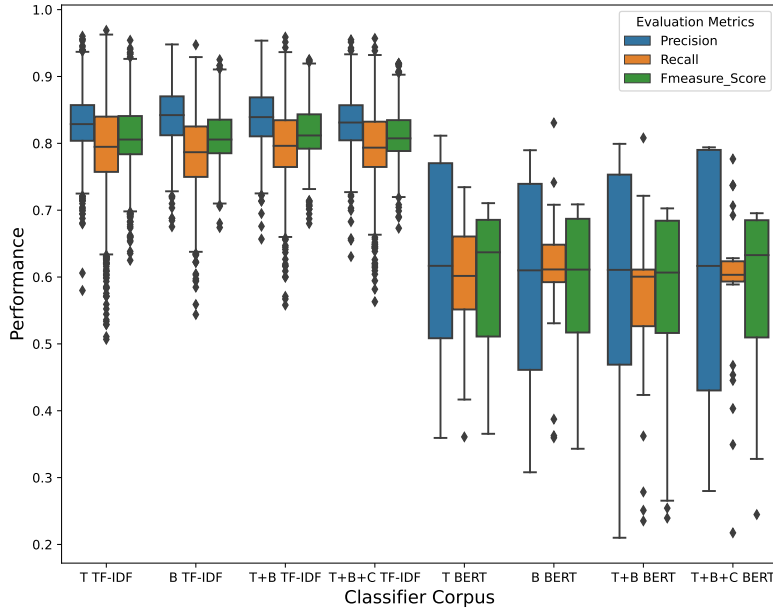
Fig. 9 Comparison between the corpus models inputted to TF-IDF and BERT. T=Title, B=Body, C=Comments

call and F-measure. However, the results suggest that using only the BODY would provide good enough outcomes since there was a negligible effect size compared to the other two configurations—using TITLE and/or COMMENTS in addition to the BODY—achieving similar results with less effort. Table 9 shows the Cliff's-delta comparison between each pair of corpus configurations, and Figure 9 shows the box plots confirming the similar results carried out by the three diverse setups. For BERT, all the models had the same distribution in precision and F-measure.

Next, we investigated the use of bigrams, trigrams, and quadrigrams, comparing the results to the use of unigrams. We used the corpus with only the issue BODY for this analysis, since this configuration was chosen in the previous step. Table 18 (Appendix A) and Figure 10 present how the algorithms perform for each n-gram configuration. While the unigram configuration has a slightly better F-measure, the quadrigram has slightly better precision. However, their differences in the precision have a negligible effect size, and their differences in F-measure have a small effect size. Additionally, the unigram uses less computational effort and memory [64]. Hence, we kept the unigram as the best option.

To investigate the influence of the machine learning (ML) classifier, we compared several options using the BODY with unigrams as a corpus. The options included: Random Forest (RF), Neural Network Multilayer Perceptron

Table 10 Cliff's Delta for F-Measure and precision: Comparison between n-grams models

n-Grams Comparison	Cliff's delta			
	F-measure		**Precision**	
1 versus 2	0.09	negligible***	-0.02	negligible**
1 versus 3	0.11	negligible***	-0.01	negligible
1 versus 4	0.15	small***	-0.06	negligible
2 versus 3	0.02	negligible	0.01	negligible***
2 versus 4	0.06	negligible***	-0.04	negligible**
3 versus 4	0.04	negligible**	-0.05	negligible***

$* p \leq 0.05;$ $** p \leq 0.01;$ $*** p \leq 0.001$
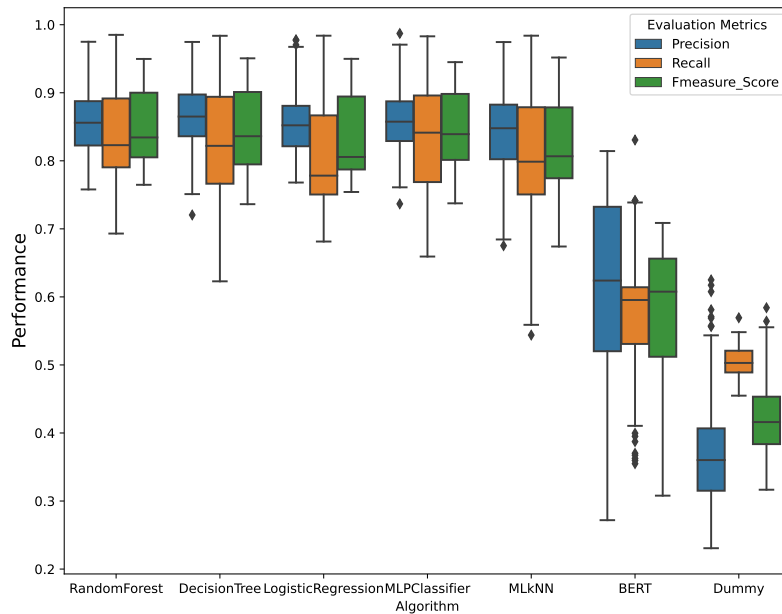
Fig. 10 Performance comparison between the machine learning algorithms

(MLPC), Decision Tree (DT), LR, MlKNN, BERT, and a Dummy Classifier with strategy "uniform." Dummy or random classifiers are often used as a baseline [65, 66]. We used the implementation from the Python package scikit-learn[9]. Figure 10 and Table 19 (Appendix A) show the comparison among the algorithms, and Table 11 presents the pair-wise statistical results comparing F-measure and precision using Cliff's delta.

Random Forest (RF) was the best model when compared to Decision Tree (DT), Logistic Regression (LR), Neural Network Multilayer Perceptron (MLPC), MlKNN algorithms, and BERT. Random Forest outperformed these

---

[9] https://scikit-learn.org/

Table 11 Cliff's Delta for F-Measure and precision: Comparison between machine learning algorithms

Algorithms	Cliff's delta			
Comparison	F-measure		Precision	
RF versus LR	0.27	small***	0.06	negligible*
RF versus MLPC	0.02	negligible	0.009	negligible
RF versus DT	0.06	negligible*	0.09	negligible***
RF versus MlkNN	0.28	small***	0.13	negligible***
RF versus BERT	1.0	large***	1.0	large***
LR versus MLPC	-0.21	small***	-0.07	negligible*
LR versus DT	-0.15	small***	-0.15	small***
LR versus MlkNN	0.07	negligible*	0.08	negligible*
LR versus BERT	1.0	large***	1.0	large***
MPLC versus DT	0.03	negligible	-0.08	negligible***
MPLC vs. MlkNN	0.24	small***	0.13	negligible***
MLPC versus BERT	1.0	large***	1.0	large***
MlkNN versus DT	-0.19	small***	-0.20	small***
MlkNN versus BERT***	1.0	large	1.0	large***
DT versus BERT***	1.0	large	1.0	large***
RF versus Dummy	1.0	large***	0.50	large***

*$p \leq 0.05$; ** $p \leq 0.01$; *** $p \leq 0.001$*

five algorithms with negligible/small effect sizes considering F-measure and precision. Compared to BERT and the Dummy Classifier, the effect size was large. The observed difference among some algorithms are fairly small and therefore might vary according to project corpus properties.

The results showed the classifier is suitable for predicting labels in projects written in different programming languages (C++, C#, and Java), with issues with vocabulary in English and Portuguese.

RQ.2.1 Summary. It is possible to individually predict the API-domain labels for each project with a precision of 0.864, recall of 0.786, and F-measure of 0.811 using the Random Forest algorithm, BODY as the corpus, and unigrams.

RQ.2.2: To what extent can we automatically attribute API-domain labels to issues using data from other projects?

Next, we merged the datasets that use English vocabulary (RTTS, JabRef, Audacity, and PowerToys), predicting the API-domain labels for all the projects. Removing the project with Portuguese vocabulary was necessary since the BERT model was trained with English vocabulary. The predictions were carried out with BODY as the corpus (and unigrams for the TF-IDF). Figure 11 shows the performance obtained with diverse algorithms. RF still had the best precision while the MLPC had the best F-measure; BERT had better precision than MLkNN and better recall than Logistic Regression. BERT was less impacted by the loss of metrics when predicting the API-domain labels with the all-projects combined dataset (Table 20 - Appendix A).
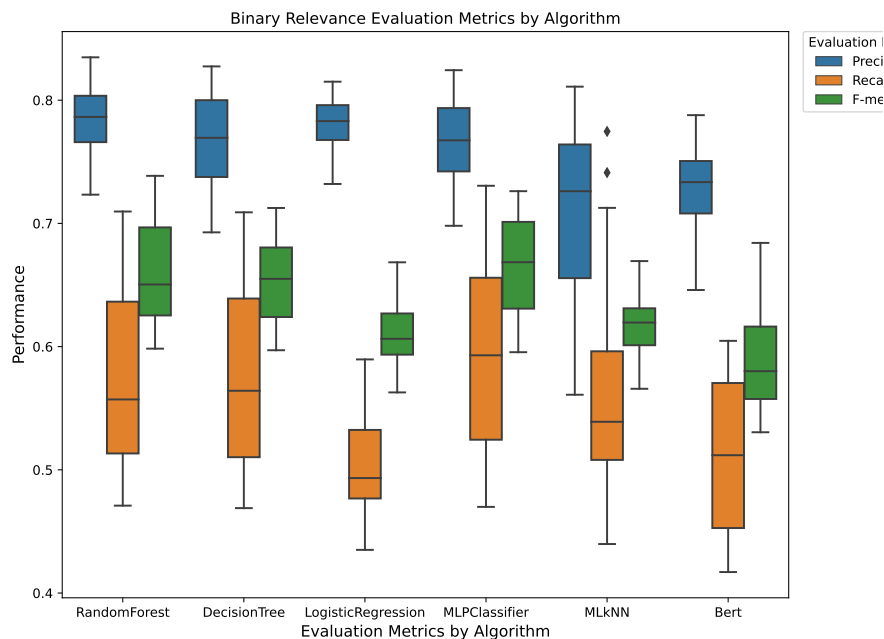
Fig. 11 Performance comparison between machine learning algorithms using the dataset with all projects - Vocabulary: EN

> **RQ.2.2 Summary.** Predicting using a dataset with all English-language projects combined decreased the precision by 9.15% using Random Forest and increased the precision using BERT by 20.63%.

RQ.2.3: To what extent can we automatically attribute API-domain labels to issues using transfer learning?

Table 12 Overall performance from models created to evaluate the transfer learning.

Training	Test	P	R	F
RTTS, Audacity, PowerToys	JabRef	0.305	0.294	0.299
JabRef, Audacity, PowerToys	RTTS	0.713	0.175	0.281
JabRef, RTTS, PowerToys	Audacity	0.688	0.284	0.402
JabRef, RTTS, Audacity	PowerToys	0.296	0.525	0.379
JabRef, Audacity, PowerToys	RTTS*	0.718	0.272	0.394

* RTTS - labels most important to the users (Section 6.2)

Finally, Table 12 shows the results for all combinations tested with transfer learning. The results had a significant range in precision and recall varying

from 0.713 to 0.296 in precision predicting RTTS and PowerToys, respectively, and from 0.525 to 0.175 in recall predicting Audacity and RTTS, respectively.

Additionally, we ran a transfer learning experiment targeting the RTTS project labels evaluated by developers (Section 6.2). We dropped all labels with fewer than three evaluations and up to 50% of "Not Important" evaluations (see Figure 12). Therefore, in the RTTS project, the labels that persisted are: "Network", "Logging", "Setup", "Micro/services", and "UI". Since Audacity, JabRef, and PowerToys projects were not evaluated by developers (Section 6), they were not included in this experiment. We observed a small increase in precision (0.713 to 0.718) and a significant increase in recall (9.7% - 0.175 to 0.272) and F-measure (11.3% - 0.281 to 0.394) - Table 12.

> ***RQ.2.3 Summary.*** Transferring learning with diverse configurations considering source and target projects decreased the metrics from 15.12% to 64.74% and the recall from 33.21% to 77.74%, depending on the sources and target project. Evaluating the transfer learning concerning only the API-domain labels evaluated as important by the developer who solved the issues improved the recall by 9.7% and F-measure by 11.3%.

6 RQ3 - Evaluating the API-domain labels with developers

Considering human input is very relevant in machine learning studies, we labeled some issues and presented them to developers that solved the same issues previously to receive feedback about how useful the API-domain labels could be if available at the time they worked on the issues.

6.1 Method

To answer RQ3, we use the Random Forest algorithm, issue description BODY as the corpus, and unigrams (the best configuration we found in RQ.2) to generate labels for the issues.

6.1.1 Labels generation

We predicted labels for 91 issues (PowerToys = 21, Audacity = 18, Cronos = 24, and RTTS = 28). The predictions covered all the 29 proposed API-domain labels (Cloud and ML do not have samples in our projects). We selected the most recently closed issues from the projects to get better chances of finding the developer who fixed the issues and they recall the problem solved. However, some issues had to be discarded when the contributor who solved them was not working for the enterprise anymore or when the OSS contributors did not answer our contact (Section 6.1.2). The use of the most recent issues and the availability of the participants created an unbalanced set of labels for

evaluation and we use our best effort to include the most representative set of API-domain labels possible in the empirical experiment.

6.1.2 Contributors assessment

In this step, we recruited 20 participants (PowerToys (1), Cronos (13), and RTTS (6)). To recruit participants from those projects we sent emails to maintainers from PowerToys and Audacity and contacted development managers from Cronos and RTTS. We asked participants from those projects to evaluate if the labels represent the skills needed to solve the issues and could help newcomers or experienced developers who want to choose an issue. All of the participants were experts in their project and were asked to evaluate the issues to which they contributed in the past. Indeed, the number of issues evaluated by participants varied according to their past contributions. Each issue was evaluated by only one participant. The participants received a gift card as a token of appreciation for their participation.

We asked the following questions:

- How important do you consider having these labels on the issue to help new contributors identify the skills needed to solve them? (Evaluate each label) (Likert: Very Important, Important, Moderately Important, Slightly Important, Not Important)
- Why?
- What labels are missing?

6.1.3 Analysis

Based on the data gathered in the contributors' assessment, we performed a quantitative analysis to assess the generated labels. To analyze the open questions in which the contributors could explain their opinions about the labels generated, we employed open coding and axial coding procedures [67].

6.2 Results

From the 91 issues predicted, we received 29.67% feedback (26 issues and 16 different API-domain labels). We did not receive feedback from the Audacity contributors. PowerToys had only one contributor who evaluated only three issues encompassing only four labels. Due to insufficient data, we removed this project from the results.

Cronos. A total of 13 contributors assessed the generated labels. Based on the results (Figure 12), the contributors described 5 labels (i.e., DevOps, UI, DB, Lang, and Security) as very important or important and APM, Setup, NLP, and IO labels unimportant. DevOps, UI, DB, and Lang were highly rated as important, with many "Very important" and "Important" evaluations. Not all the participants justified their response, but among the reasons
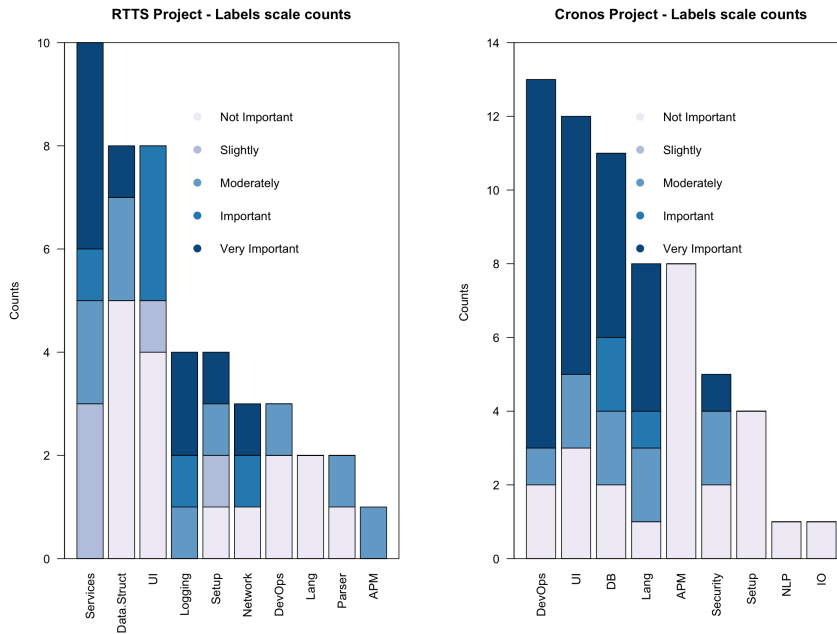
Fig. 12 Labels assessment by project

those contributors mentioned that "*It was a simple UI issue.*" (P15) indicating the success of the "UI" prediction. Another developer mentioned "*This issue also required database, logic and lang skills.*" (P6). This issue was tagged with "UI" and "DevOps" (evaluated as "Very important") but the developer missed some skills. Related to the missing labels, some contributors mentioned that "*The issue is related to a restriction. It requires UI skills and DevOps skills (not included in the predictions. [...]But it also requires other skills.*" (P18). Another contributor also missed some labels and mentioned "*This issue also required database, logic, and language skills.*" (P06).

RTTS. Concerning the RTTS project, five contributors assessed the labels generated for the issues they had solved; in this scenario, we have contributors evaluating from 1 to 3 issues each. Our findings (Figure 12) highlight that the following labels were classified by contributors as important to very important: Services, UI, Logging, Setup, Network, and Data Structure. Among the reasons contributors highlighted, we can observe the positive feedback as mentioned in: "*I can totally agree with the labels for this, as to find the problem and apply the solution all the skills are necessary.*" (P20). Moreover, contributors classified the following labels as not important or slightly important: Lang, Parser, DevOps, UI, and Data Structure. Some contributors mentioned that: "*I partially agree, some of the labels could give an initial point of view to the reported issue, but some are not related, like Language, Data structure and Setup*" (P2). Some contributors reported missing some labels according

to what was mentioned: "*Logging skills would be necessary to troubleshoot the issue and get the relevant information from the application, while service skills (knowledge about how service discovery and the service registry in the system works) would be necessary to find that the service version of the requested service didn't match what was registered. As for Network, it could have been useful to be able to determine that this issue was not caused by some error/faulty response from the requested service, but in this case, the log stated explicitly that the requested service did not exist. I don't find that the other labels/skills apply to this issue.*" (P04).

RQ.3. Summary. Our findings suggest the labels would be useful to help identify the skills needed to solve them. The efficiency of labels generated differs by project, for Cronos we had 61.9% of the labels evaluated in the range from slightly to Very important and 64.4% in the RTTS project in the same range.

---

7 Discussion

This section discusses our results and future work.

Do developers have a well-defined preference about labels? The feedback shared by study participants in Section 4 showed us the importance of the different types of labels to ease the issue selection process. However, the developers expressed preferences about different types of labels, and some preferences are ambiguous. For instance, P32 indicated "The technology used" when we asked what kind of labels they want to see in the issues. The technology could refer to a "programming language," or an "API." While both classifications could be used, we would prefer to define the technology as API because it is more specific than a programming language or even a framework that can encompass many libraries. A similar situation emerged with "priority" (P12). The "priority" could be restricted only to "low" or "high," or could it include other aspects like the "impact on operations", as suggested by P15.

We can group the kind of labels in technology (technology, API, programming language, database, framework, and architecture layer) and management (type, priority, status, difficulty level, GitHub info). Management labels are more often used in issue trackers. In this work, we propose to add to the issues a kind of technological label, the API-domain labels, which we claim are a proxy for the skills needed to solve an issue. Nonetheless, one should avoid overloading the issue trackers with too many labels. Future research can investigate the right balance of offering labels without creating a visual overhead for the contributor.

Are API-domain labels relevant? Our findings show that participants considered API-domain labels relevant in selecting issues. More specifically, newcomers to the projects considered API-domain labels more relevant than other general labels that describe the components and slightly more favored

than management labels describing the type of issue. This suggests that a higher-level understanding of the API domain is more relevant than deeper information about the specific component in the project.

When controlling for issue type and component, API-domain labels were considered more relevant for experienced coders than novices (or students). This suggests that novices may need more help than "just" the technology for which they need skills. Our results also show that novices could be helped if the issues provide additional details about the complexity levels, how much knowledge about the particular APIs is needed, the required/recommended academic courses needed for the skill level, estimated time to completion, contact for help, etc.

Although each contributor is a newcomer when they move to a new project, previous experience counts when the new project shares technology with the previous projects. As opposed to experienced newcomers, who may transfer knowledge from previous projects and jump directly to the issue solution, novice newcomers spend more time understanding the project structure, the underlying technology, and how to set up the environment [11] which might suggest why practitioners from the industry and experienced participants selected more API-domain labels than students and novices. Perhaps the API granularity is deeper than what the novices are looking for. Future research may consider the appropriate technical information to assist novice newcomers.

In addition to API-domain labels, what issue characteristics are relevant to identify skills in issues? In addition to labels, new contributors mentioned the TITLE, BODY, and COMMENTS as sources of information to identify the necessary skills to work on the issues. Such elements can be structured with issue templates or written in an ad-hoc manner. Santos et al. [11] asked maintainers to suggest community strategies to help newcomers find a suitable issue. Among the identified strategies, maintainers suggested 15 diverse ways of labeling the issues (e.g., labeling with skills, knowledge area, programming languages, libraries, and others) and several ways of organizing the issues, which include creating templates.

While these other issue elements may indicate the skills and other characteristics of the issues that are not on the labels, some issues – and existing templates – are incomplete, lacking important information for contributors. The 5W2H analysis we applied in this paper can help us to holistically understand what should be written in issues by covering the seven dimensions of information - who, why, when, what, where, how to solve and how big is the issue. Future work can use the 5W2H questions to inspect the completeness of existing templates in terms of covering the seven dimensions of information.

Despite the importance of issue templates, we removed template sentences in an effort to clean repeated text to be ingested by the data processing pipeline. For example, one template sentence is "Steps to reproduce." Since this fixed text appears in many issues (regardless of their categories) and the templates had changed over time, we decided to remove it before processing the issue corpus. This removal only affects the trained model, and we still should

use the results of the 5W2H analysis to create a human-oriented template able to point new contributors to information relevant to them.

What are the effects of the corpus characteristics on the labels' classifications? Observing the reported results (TF-IDF) for different corpora used as input, we noticed that the model created using only the issue body performed similarly to the models using the issue title, body, and comments, and better than the model using only the title. By inspecting the results, we noticed that by adding more words to create the model, the matrix of features becomes sparse and does not improve the classifier's performance.

We also found co-occurrence among labels. For instance, "Test", "Logging", and "i18n" appeared often together (Figure 13). This is due to the strong relationship found in the source files. By searching the references for these API-domain categories in the source code, we found "Test" in 4,579 source code files, compared to "Logging" in 903. The label "i18n" appeared in only 73 files. On the other hand, the API-domain labels for "CG" and "Security" usually do not co-occur. "CG" only appeared in five java files, while "Security" appeared in only 47 files. Future research can investigate co-occurrence techniques to predict co-changes in software artifacts (e.g., [68]) in this context.

Figure 13 exhibits the labels' co-occurrence for the dataset containing all the projects. A co-occurrence matrix presents the number of times each label appears in the same context as each possible other label. Examining the aforementioned co-occurrence data, we can determine some expectations and induce some predictions. For example, the "DB" label (Database) occurred with more frequency alongside "Network" and "Thread." So, it is possible to guess when an issue has both labels, and we likely can suggest a "Database" label, even when the machine learning algorithm could not predict it. A possible future work can combine the machine learning algorithm proposed in this work with frequent itemset mining techniques, such as apriori [69].

What are the difficulties in labeling accurately? We suspect that the high occurrence of "UI", "Util", and "Logic" labels (> 500 issues) compared with the low occurrence of "i18n", "Interpreter", "GIS", and "NLP" (< 57 issues) may influence the precision and F-measure values. We tested the classifier with only the top 5 most prevalent API-domain labels and observed no statistically significant differences. One possible explanation is that the transformation method used to create the classifier was Binary Relevance, which creates a single classifier for each label and overlooks possible co-occurrence.

The dataset is unbalanced due to the characteristics of the projects. Since JabRef, for instance, is a desktop application, the API-domain label "UI" appears more frequently. Table 13 shows the confusion matrix for the dataset containing all projects (for individual projects, see the appendix). This impacts the prediction of the minor labels even with the SMOTE algorithm, which improves the occurrences of rare labels. Some labels only appear in a few projects. Therefore, even when they are common in a specific project when training and testing with all projects, they may become rare. The recommendation of labels with poor results should be avoided because of the risk of indicating a wrong skill to the contributor.
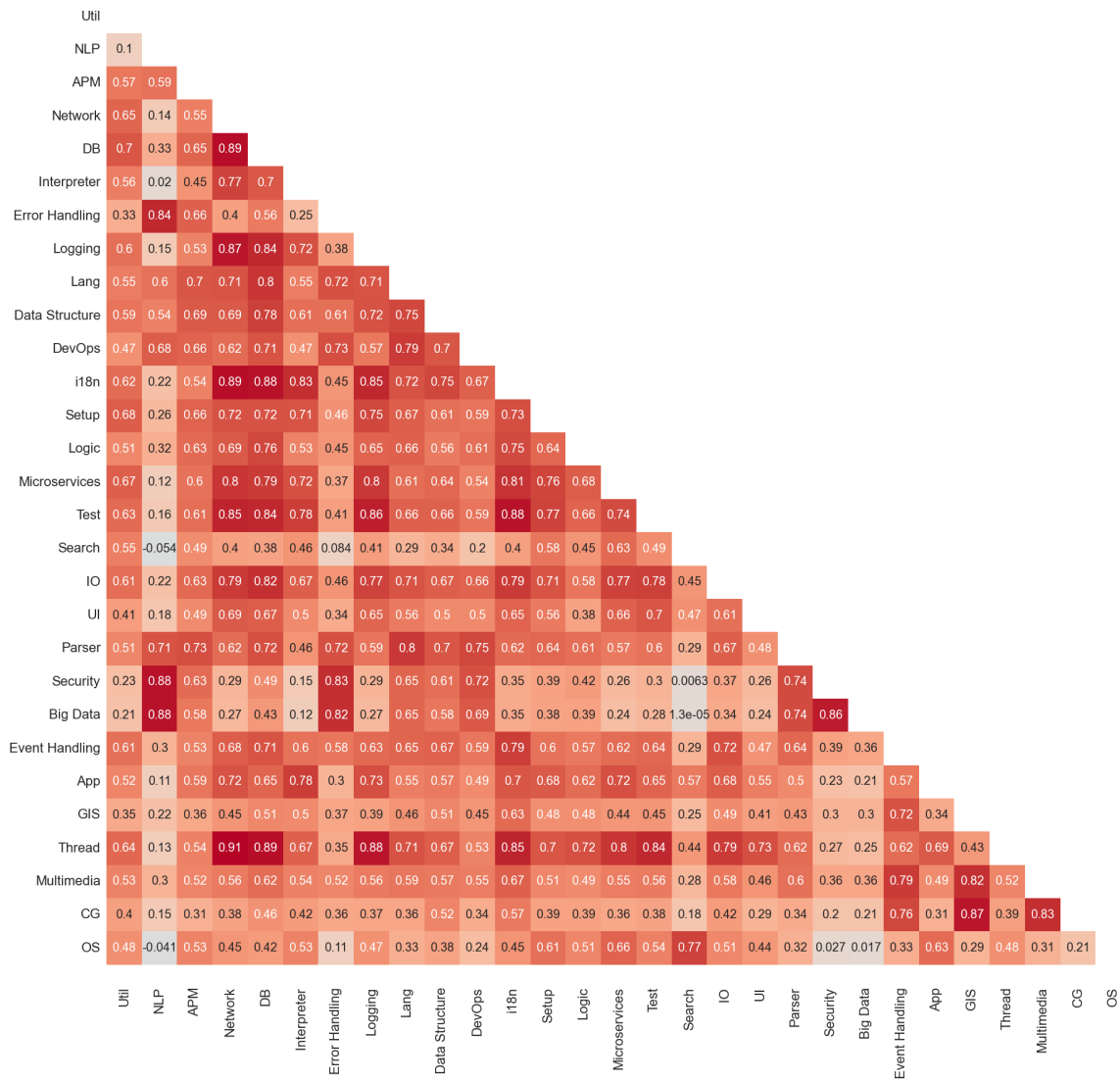
Fig. 13 Heat Map - Label correlation in the dataset with all projects combined. The darker, the more correlation exists between the labels.

Table 13 Confusion matrix data and performance from the selected model with all projects

API-domain	TN	FP	FN	TP	Precision	Recall
APM	125	4	44	8	0.66	0.15
App	80	22	19	60	0.73	0.75
Big Data	152	0	29	0	0	0
Data Structure	78	24	6	73	0.75	0.92
DB	163	2	11	5	0.71	0.31
DevOps	113	26	0	42	0.61	1
Error Handling	97	32	5	47	0.59	0.90
Event Handling	162	1	8	10	0.9	0.55
GIS	178	2	0	1	0.33	1
Interpreter	173	2	3	3	0.6	0.5
IO	141	8	5	27	0.77	0.84
i18n	166	7	5	3	0.3	0.375
Lang	112	36	0	33	0.47	1
Logging	174	1	4	2	0.66	0.33
Logic	68	9	2	102	0.91	0.98
Micro/services	151	1	23	6	0.85	0.2
Network	175	0	6	0	0	0
NLP	164	0	17	0	0	0
OS	119	9	8	45	0.83	0.84
Parser	101	28	3	49	0.63	0.94
Search	134	9	15	23	0.71	0.6
Security	151	0	30	0	0	0
Setup	38	56	9	78	0.58	0.89
Test	166	0	15	0	0	0
UI	10	33	3	135	0.8	0.97
Util	84	4	16	77	0.95	0.82
Total	3275	316	286	829		

Despite the lack of accuracy in predicting the rare labels, we were able
to predict those with more than 200 occurrences (all projects together) with
reasonable precision (0.84) and/or recall (0.78). We argue the project's nature
contributes to the number of issues related to their domain. For example,
since the Audacity project is an audio editor and recorder, a high occurrence
of "UI", "IO", and "Multimedia" labels is expected. We argue that Audacity's
nature contributes to the number of issues related to the labels above. Labels
with few samples suffered from low or unstable metrics. "DB", for example,
varied from 0.09 to 0.9 in recall on predictions depending on the text/train
split.

Improving the performance of BERT. In addition to the number of
occurrences of a label, the BERT metrics can be improved by increasing the
training set size. Wang et al. [29] and their exploration of several trained deep
learning models for GitHub labeling provide important insights into poten-
tial performance increases with BERT. The authors showed that the BERT
model performed better than the other language models for large datasets
with at least 5,000 issues, achieving the highest accuracy, precision, recall,
and F-measure scores. However, for small datasets with less than 5,000 is-
sues, CNN outperformed BERT as the best model overall. This suggests that
BERT depends on the size of the training set of corpus data. Therefore, the

performance of BERT when labeling GitHub issues will improve with an increased dataset size for the targeted open-source project. When the project datasets were merged (Table 11), the BERT metrics decreased the difference from about 26% to 6% in precision compared to the other classifiers.

What is the impact of the expert classification? Experts can also help increase the classification metrics for all models. We could observe the C++ project achieved the best F-measure compared with the Java and C# projects (0.84, 0.82, and 0.80, respectively, with small to large effect sizes). Although we evaluated only one C++ project, the results might suggest after examining Table 8 that the number of APIs evaluated by the experts impacts the metrics we will obtain. On the other hand, manual evaluation of a high number of APIs may make generalization unfeasible. The classification carried out by the experts in the C++ project comprised a higher percentage of APIs analyzed. This might be caused by the language characteristics: the libraries' names parsed from the C++ source code had limited information about their use. Thus, classification was more time-consuming. Indeed, the C++ project demanded more effort from the experts to classify it. Ultimately, it became a more detailed classification with better prediction metrics.

While experts' analyses are time-consuming, some outlier projects require much less effort than others. For instance, experts analyzed fewer than 3% of the APIs in RTTS. Since this project imports popular libraries, reuses many libraries across the entire source code and is modular, the expert's work was easier. A possible relationship between popular APIs, modularization, and expert evaluation should be explored in future work. Another possible future work should identify what programming language characteristics impact the expert classification.

To what extent does the proposed method generalize? The semi-automatic classification process decreased the effort carried out by the experts to define the expertise of the APIs. Despite there being considerable effort remaining, as the dataset increases, the rate of new APIs to classify should decrease since projects reuse an average of 35-53% of core APIs. Third-party libraries account for 8-32% and 45% on average (Core + third-party). The use of popular open-source APIs could lead to an impressive 85% of shared APIs between projects [70]. Farther, the project sizes grow much more quickly than the size of uniquely-used API entities [70].

Thus, the demand for expert evaluation should decrease significantly when the number of mined libraries reaches a critical mass (for each programming language), and even new projects may use previous expert evaluations. This might impact the method when applied to industry projects, which may use a variety of unique non-free APIs. However, API sharing may happen inside companies or business units, repeating the phenomenon of the libraries' critical mass. Nevertheless, we did not observe this effect, and we could predict labels for an OSS project using data from an industry project.

To what extent does the model perform transfer learning? Transfer learning is crucial when projects lack data for training (cold start) or the time or infrastructure to develop their own models. This can be particularly

problematic in the industry since the data can have restricted access due to security precautions or to comply with procedures or laws. In this situation, the ability to use a pre-trained model is necessary. Using pre-trained models to predict from new data is also desirable because it is faster and cheaper than retraining a model every time a new source project is added to the dataset [15, 71]. The projects may also benefit from the complementary data from another project when the project dataset is too small for training a predictive model.

The transfer learning experiments found a decrease in precision and recall. The metrics definition: $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$ indicates the number of False Negatives (FN) and False Positives (FP) that should impact the results. For example, in training and testing individual projects, the RTTS project had a small number of PFs and FNs compared to the transfer learning experiment when RTTS was a target project (Tables 14 and 15). When targeting the RTTS project, the high number of FNs significantly decreased the recall metric. On the other hand, targeting PowerToys, the number of FPs negatively impacted the precision (Tables 15, 22, and 12). The projects only shared a small number of labels (5 in 31) and are imbalanced among the datasets. For example, "Setup" is popular in the RTTS project and rare in JabRef, suggesting the conditional probability distribution of the sources and targets differ. These characteristics might determine which projects match and, therefore, be used to decide the transfer learning source or target. Future work should investigate whether the domain, platform (Web, Desktop, Mobile), architecture, or other project property derives a good match. Furthermore, investigating proxy techniques, such as the one proposed by Nam et al. [15], to minimize the data distribution difference between target and source projects to predict software engineering defects can be applied to predictions of domain labels of API. Results for the JabRef (Table 25) and Audacity (Table 26) projects using transfer learning are available in Appendix A. We can observe the high number of FP and FN comparing the Audacity transfer learning results in Table 26 and the results of training and testing the Audacity dataset alone (Table 23). Similarly, we can observe the same pattern in the JabRef results in Tables 21 and 25.

How did the contributors rate the labels generated for the issues they solved?

Overall, participants evaluated the generated labels with positive feedback. The labels classified as important or very important across all the projects were: DevOps (10), DB (5), Services (4), UI (14), Lang (5), Security (1), and Logging (3). Moreover, the labels that were classified as not important were: APM (8), Setup (7), Data Structure (6) and UI (9), and Security (2).

In the RTTS project, all four best-evaluated labels (Services, Logging, Setup, and Network) had precision above 0.75, and two of the four worst-evaluated ones (Data Structure, UI, DevOps, and Lang) had precision ≤ 0.7. A threshold could determine whether a label must be reported.

Participants from Cronos projects mentioned they would like to see the label "Data Structure" for the evaluated issues. This occurred because we removed the label Data Structure once it was generated for 90% of the issues

Table 14 Confusion matrix and performance. Project RTTS trained/tested alone

API-domain	TN	FP	FN	TP	Precision	Recall	F-measure
APM	112	4	2	24	0.85	0.92	0.88
Big Data	134	1	2	5	0.83	0.71	0.76
Data Structure	25	36	3	78	0.68	0.96	0.80
DB	84	4	19	35	0.89	0.64	0.75
DevOps	60	17	13	52	0.75	0.80	0.77
Error Handling	126	5	4	7	0.58	0.63	0.60
Event Handling	129	0	2	11	1	0.84	0.91
i18n	121	6	7	8	0.57	0.53	0.55
Lang	28	29	11	74	0.71	0.87	0.78
Logging	47	24	18	53	0.68	0.74	0.71
Microservices	1	12	0	129	0.91	1	0.95
Network	65	20	21	36	0.64	0.63	0.63
Parser	69	17	14	42	0.71	0.75	0.73
Security	129	0	8	5	1	0.38	0.55
Setup	66	10	33	33	0.76	0.50	0.60
UI	4	15	0	123	0.89	1	0.94
Total	1200	200	157	715			

Table 15 Confusion matrix and performance: Project RTTS - transfer learning.

API-domain	TN	FP	FN	TP	Precision	Recall	F-measure
APM	197	0	38	0	0	0	0
Data Structure	100	0	135	0	0	0	0
DB	145	0	90	0	0	0	0
Error Handling	223	0	12	0	0	0	0
Event Handling	212	0	23	0	0	0	0
Lang	114	0	121	0	0	0	0
IO	44	21	131	39	0.65	0.22	0.33
i18n	214	0	21	0	0	0	0
Logging	102	22	91	20	0.47	0.18	0.26
Logic	13	17	146	59	0.77	0.28	0.41
Microservices	28	0	206	1	1	0.004	0.009
Network	129	0	106	0	0	0	0
Parser	156	0	79	0	0	0	0
Setup	115	14	98	8	0.36	0.07	0.12
Thread	175	0	60	0	0	0	0
UI	3	38	26	168	0.81	0.86	0.84
Total	1970	112	1383	295			

selected in the Cronos project. One possibility for that case would be to include in the description of the project that it is strongly based on data structures and that the reported issues likely would involve this knowledge.

In addition, participants reported some labels could provide a clue for looking for the bug's root cause or determining the work needed to address a new feature request. For Example: *"...some of the labels could give an initial point of view to the reported issue"* (P2) or *"Network: While network tag wasn't that necessary for this particular case, the issue could have been caused by a communication error between the services in which case they would have been"* (P4). On the other hand, some participants preferred not to see more general

Table 16 Confusion matrix and performance: Project PowerToys - transfer learning.

API-domain	TN	FP	FN	TP	Precision	Recall	F-measure
APM	264	1	88	0	0	0	0
App	315	9	28	1	0.66	0.76	0.71
Data Structure	344	3	6	0	0.03	0.40	0.06
i18n	209	134	4	6	0	0	0
Interpreter	342	1	10	0	0	0	0
Logging	153	196	0	4	0.007	0.25	0.01
Logic	173	27	100	53	0.01	1	0.02
Microservices	0	348	0	5	0.25	0.49	0.33
Parser	6	105	10	232	0.07	0.13	0.09
Setup	351	0	2	0	0.50	0.07	0.13
Test	174	82	56	41	0.006	0.66	0.01
UI	105	245	1	2	0.68	0.92	0.78

labels, like Data Structure or Logging, since they are present in many issues: *"Data Structure is literally everywhere, there wouldn't be any program without them"* (P1), while others missed the Data Structure label (not present in the predicted list because it reached the 90% threshold) and suggested including it (P14, P16, and P17). Future work can determine how to address developer preferences regarding the inclusion of general labels.

The generalization of the method proposed in this paper assisted us in embracing more projects. Nevertheless, it also brought problems. We proposed generic labels able to fit a wide range of project types. This might explain the comments about the generic labels. *"...It was a backward compatibility issue with user-defined configuration data, so with a generous interpretation Setup was accurate, but I would have preferred Information Model or Domain Model had it existed"* (P1). Analyzing the participant's suggestion for a "Validation" label, we recollect to the point where the NLP similarity suggested possible API domains for the library related to the issue and the experts' choice. We found the selected API domain was "Logic" since no "Validation" API-domain label was available. If the experts came from the project, perhaps the API-domain label "Validation" could be present and thus meet the participant's needs.

Future work can explore more API-domain labels to expand and propose more options to fit additional projects. Customizing labels for the project may generate more precise directions about the skills needed but will require more expert work time. On the other hand, generalization expands the method to a huge range of projects and can decrease the meaning level of the API domains.

What are the practical implications for different stakeholders? ***New contributors.*** API-domain labels can help open-source contributors, enabling them to review the skills needed to work on the issues upfront. This is especially useful for new contributors and casual contributors [72, 73], who have no previous experience with the project terminology.

Project maintainers. Automatic API-domain labeling can help maintainers distribute team effort to address project tasks based on required expertise.

Project maintainers can also identify which type of APIs generate more issues. Our results show that we can predict the most prominent API domains—in this case "Util" and "Logic"— with precision up to 95% and 91%, respectively (see Table 13).

Platform/Forge Managers. Participants often selected TITLE, BODY, and LABELS to look for information when choosing an issue to which to contribute. Our results can be used to propose better layouts for the issue list and detail pages, prioritizing them against other information regions (2). In the issue detail page on GitHub, for instance, the label information appears outside of the main contributor focus, on the right side of the screen.

Templates to guide GitHub users in filling out the issues' body to create patterns can be useful in not only making the information space consistent across issues, but also helping automated classifiers that use the information to predict API labels. For instance, some of the wrong predictions in our study could be caused by titles and bodies with little useful information from which to generate labels.

Researchers. The scientific community can extend the proposed approach to other languages and projects, including those with more data and different algorithms. Our approach can also be used to improve tools that recommend tasks matched to new contributors' skills and career goals (e.g., [74]).

Educators. Educators who assign contributions to OSS as part of their coursework [75] can also benefit from our approach. Labeling issues in OSS projects can help them select examples or tasks for their classes, bringing a practical perspective to the learning environment.

8 Threats to Validity

The threats to validity are divided into "internal," "construct," and "external."

Internal Validity. One of the threats to the validity of this study is the API domain categorization. We acknowledge the threat that different individuals can create different categorizations, which may introduce some bias in our results. To mitigate this problem, three individuals, including two senior developers and a contributor to the JabRef project, created the API-domain labels categories aiming to generalize to any type of project. In the future, we can improve this classification process with a collaborative approach (e.g., [76, 77]).

Although participants with different profiles participated in the JabRef user study, the sample cannot represent the entire population, and the results can be biased. The study randomly assigned a group to each participant. However, some participants did not finish the questionnaire, and the groups ended up lacking balance. Also, the way we created subgroups can introduce bias in the analysis. The practitioners' classification as industry and students were done based on the location of the recruitment, and some students could also be industry practitioners and vice-versa. However, the results of this analysis were corroborated by aggregation according to experience level.

Construct Validity. Another concern is the number of issues in our dataset and the link between issues and pull requests. To include an issue/key/-tracking ID in the dataset, we linked it to its solution submitted via pull request (or "revision" and "trouble id"). By linking them, we could identify the APIs used to create the labels and define our ground truth (check Section 5.1.1). This study does not identify issues merged without PR information. We manually inspected a random sample of issues (or "keys" and "tracking ids") to check whether the data was correctly collected and reflected what was shown on the ITS interface. Two authors manually examined 100 tasks randomly picked up from the projects, comparing the collected data with the GitHub interface. All records were consistent, and all of the issues in this validation set were correctly linked to their pull requests. When the linked data had more than one correspondence, we concatenated all data using the appropriated corpus entry (title, body, comments, description, and summary). Some of the linked data occasionally had repeated text, and can overfit our model. Future versions may improve the data cleaning step. Unlike the other projects, Cronos had multiple linked data through the following columns: "pai" and "ramo," "linked issue" and "key," and "key" and "ramo." This creates a recursive situation where we may link each update with many "keys" in different ways. We preferred to keep it simple, using only the linked data that was similar to the other projects: "key" and "ramo."

In prediction models, overfitting occurs when a prediction model exhibits random error or noise instead of an underlying relationship. During the model training phase, the algorithm used information not included in the test set. To mitigate this problem, we also used a shuffle method to randomize the training and test samples.

Further, we acknowledge that we did not investigate whether the labels helped the users find the most appropriate tasks. It was not part of the user study to evaluate how effective the API labels were in finding a match with user skills. Our focus was on understanding the relevance that the API-domain labels have on the participants' decisions. Besides, we did not evaluate how false positive labels would impact task selection or ranking. However, we believe the impact is minimal since in the three most selected issues, out of 11 recommendations in the JabRef project, only one label was a false positive. In addition, when we asked the participants to pick issues with the API labels + project labels (treatment group) or project labels (control group), we might introduce some bias. Indeed, evaluating the difference of relevance perception introduced by the appearance of the new (API-domain) labels should have some influence brought by the poor performance of the project's labels, masking the difference in the measurement experiment. Investigating the effectiveness of API labels by an experiment matching contributors and tasks skills and identifying the problems caused by misclassification are potential avenues for future work. The empirical experiment to pick an issue and ask the relevant regions for that choice may introduce a bias since the participant only selected an issue and did not solve the issue.

When classifying the issues and linked pull requests, we compared the files changed with the parsed source code files at the last version of the projects. If the updated source file is not present anymore, the pull request is discarded.

External Validity. Generalization is also a limitation of this study. The outcomes could differ for other projects, programming languages ecosystems, or even issues written in a different language. To address this limitation, we extended the previous study [14] in that direction, mining different projects, including three programming languages, and two natural languages (or vocabularies). Nevertheless, this study showed how a multi-label classification approach could be useful for predicting API-domain labels and how relevant such a label can be to new contributors. Moreover, the API-domain labels that we identified can generalize to other projects that use the same APIs across multiple project domains (Desktop and Web applications). Many projects adopt a typical architecture (MVC) and frameworks (JavaFX, JUnit, etc.), which makes them similar to many other projects. As described by Qiu et al. [70], projects adopt common APIs, accounting for up to 53% of the APIs used. Moreover, our data can be used as a training set for automated API-domain label generation in other projects.

9 Conclusion

We investigated whether API-domain labels are used by newcomers to select an issue and what information newcomers use to decide what issue to contribute. We found that industry practitioners and experienced coders prefer API-domain labels more often than students and novice coders. Participants prefer API-domain labels over component labels already used in the project. Users would like to see labels with information about issue type, priority, programming language, complexity, technology, and API and pick an issue based on title, body, comments, and labels.

We also investigate to what extent we can predict API-domain labels. We mined data from 22,231 issues from five projects and predicted 31 API-domain labels. Training and testing the projects separately, TF-IDF with the Random Forest algorithm (RF), and unigrams obtained a precision of 84% and overcame BERT (precision of 62%). Data from the issue body offered the best results. However, when predicting the API-domain labels for all projects together, RF precision decreased to 78%, and BERT increased to 72%, suggesting the positive sensibility of the BERT technique when applied to larger datasets.

Transferring learning from diverse sources and targets resulted in a decrease in evaluation metrics with an extensive range of values regarding precision and recall. Future work should investigate ways to determine when or how to apply transfer learning to API-domain labels among projects.

Finally, developers agreed that up to 64.4% of the API-domain labels are important to identify the skills and therefore should help to solve the issues if they are available.

This study is a step toward helping new contributors match their API skills with each task and better identify an appropriate task to start their onboarding process into an OSS project.

Acknowledgment

Data availability

The datasets generated during and/or analyzed during the current study are available in the zenodo repository[10].

Declarations

Conflicts of interest/competing interests The authors declare that they have no conflict of interest.

References

1. J. Wang and A. Sarma, "Which bug should i fix: helping new developers onboard a new project," in *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering.* ACM, 2011, pp. 76–79.
2. I. Steinmacher, T. U. Conte, and M. A. Gerosa, "Understanding and supporting the choice of an appropriate task to start with in open source software communities," in *2015 48th Hawaii International Conference on System Sciences.* IEEE, 2015, pp. 5299–5308.
3. I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," *Information and Software Technology*, vol. 59, pp. 67–85, 2015.
4. I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social barriers faced by newcomers placing their first contribution in open source software projects," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1379–1392.

---

[10] `https://doi.org/10.5281/zenodo.6869246`

5. C. Stanik, L. Montgomery, D. Martens, D. Fucci, and W. Maalej, "A simple nlp-based approach to support onboarding and retention in open source communities," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*.   IEEE, 2018, pp. 172–182.

6. T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, 2010.

7. N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, "Quality of bug reports in eclipse," in *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange*, ser. eclipse '07.   New York, NY, USA: ACM, 2007, pp. 21–25.

8. L. Vaz, I. Steinmacher, and S. Marczak, "An empirical study on task documentation in software crowdsourcing on topcoder," in *2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE)*.   IEEE, 2019, pp. 48–57.

9. I. Santos, I. Wiese, I. Steinmacher, A. Sarma, and M. A. Gerosa, "Hits and misses: Newcomers' ability to identify skills needed for OSS tasks," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, pp. 174–183.

10. I. Steinmacher, C. Treude, and M. A. Gerosa, "Let me in: Guidelines for the successful onboarding of newcomers to open source projects," *IEEE Software*, vol. 36, no. 4, pp. 41–49, 2018.

11. F. Santos, B. Trinkenreich, J. F. Nicolati Pimentel, I. Wiese, I. Steinmacher, A. Sarma, and M. Gerosa, "How to choose a task? mismatches in perspectives of newcomers and existing contributors," *Empirical Software Engineering and Measurement*, 2022.

12. A. Barcomb, K. Stol, B. Fitzgerald, and D. Riehle, "Managing episodic volunteers in free/libre/open source software communities," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

13. "Api   definition."   [Online].   Available: https://languages.oup.com/google-dictionary-en/

14. F. Santos, I. Wiese, B. Trinkenreich, I. Steinmacher, A. Sarma, and M. A. Gerosa, "Can I solve it? identifying apis required to complete OSS tasks," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*.   IEEE, 2021, pp. 346–257.

15. J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *2013 35th international conference on software engineering (ICSE)*.   IEEE, 2013, pp. 382–391.

16. M. Izadi, S. Ganji, and A. Heydarnoori, "Topic recommendation for software repositories using multi-label classification algorithms," *Empir. Softw. Eng.*, vol. 26, p. 93, 2021.

17. S. Vargas-Baldrich, M. Linares-Vásquez, and D. Poshyvanyk, "Automated tagging of software projects using bytecode and dependencies," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.   IEEE, 2015, pp. 289–294.

18. X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *2013 10th Working Conference on Mining Software Repositories (MSR).* IEEE, 2013, pp. 287–296.

19. B. Lin, F. Zampetti, G. Bavota, M. Di Penta, and M. Lanza, "Pattern-based mining of opinions in q&a websites," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE).* IEEE, 2019, pp. 548–559.

20. G. Uddin and F. Khomh, "Automatic mining of opinions expressed about apis in stack overflow," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

21. G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement? a text-based approach to classify change requests," in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, 2008, pp. 304–318.

22. N. Pingclasai, H. Hata, and K.-i. Matsumoto, "Classifying bug reports to bugs and other requests using topic modeling," in *2013 20Th asia-pacific software engineering conference (APSEC)*, vol. 2. IEEE, 2013, pp. 13–18.

23. Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 150–176, 2016.

24. Y. Zhu, M. Pan, Y. Pei, and T. Zhang, "A bug or a suggestion? an automatic way to label issues," *arXiv preprint arXiv:1909.00934*, 2019.

25. F. El Zanaty, C. Rezk, S. Lijbrink, W. van Bergen, M. Côté, and S. McIntosh, "Automatic recovery of missing issue type labels," *IEEE Software*, 2020.

26. Q. Perez, P.-A. Jean, C. Urtado, and S. Vauttier, "Bug or not bug? that is the question," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC).* IEEE, 2021, pp. 47–58.

27. R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Ticket tagger: Machine learning driven issue classification," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 2019, pp. 406–409.

28. M. Izadi, K. Akbari, and A. Heydarnoori, "Predicting the objective and priority of issue reports in software repositories," *Empirical Software Engineering*, vol. 27, no. 2, pp. 1–37, 2022.

29. J. Wang, X. Zhang, and L. Chen, "How well do pre-trained contextual language representations recommend labels for GitHub issues?" *Knowledge-Based Systems*, vol. 232, p. 107476, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705121007383

30. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019.

31. Y. Park and C. Jensen, "Beyond pretty pictures: Examining the benefits of code visualization for open source newcomers," in *Proceedings of the 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, ser. VISSOFT '09. IEEE, Sep. 2009, pp. 3–10.

32. Y. Huang, J. Wang, S. Wang, Z. Liu, D. Wang, and Q. Wang, "Characterizing and predicting good first issues," in *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2021, pp. 1–12.

33. M. Szumilas, "Explaining odds ratios," *Journal of the Canadian academy of child and adolescent psychiatry*, vol. 19, no. 3, p. 227, 2010.

34. D. Sheshin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 5th ed. Chapman & Hall, 2020.

35. T. Ohno, "How the toyota production system was created," *Japanese Economic Studies*, vol. 10, no. 4, pp. 83–101, 1982.

36. H. Pacaiova, "Analysis and identification of nonconforming products by 5W2H method," *Center for Quality*, 2015.

37. A. C. T. Klock, I. Gasparini, and M. S. Pimenta, "5W2H framework: a guide to design, develop and evaluate the user-centered gamification," in *Proceedings of the 15th Brazilian Symposium on Human Factors in Computing Systems*, 2016, pp. 1–10.

38. S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 573–591, 2009.

39. A. Savidis and C. Savaki, "Software architecture mining from source code with dependency graph clustering and visualization," in *IVAPP*, 12 2021.

40. "spacy industrial-strength natural language processing." https://spacy.io/, accessed: 2021-10-04.

41. Y. Feng, J. Jones, Z. Chen, and C. Fang, "An empirical study on software failure classification with multi-label and problem-transformation techniques," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2018, pp. 320–330.

42. T. Guggulothu and S. A. Moiz, "Code smell detection using multi-label classification approach," *Software Quality Journal*, vol. 28, no. 3, pp. 1063–1086, 2020.

43. J. Ramos *et al.*, "Using TF-IDF to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242. Piscataway, NJ, 2003, pp. 133–142.

44. S. Ravichandiran, *Getting Started with Google BERT: Build and train state-of-the-art natural language processing models using BERT*. Packt Publishing Ltd, 2021.

45. D. Behl, S. Handa, and A. Arora, "A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf," in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*. IEEE, 2014, pp. 294–299.

46. S. L. Vadlamani and O. Baysal, "Studying software developer expertise and contributions in Stack Overflow and GitHub," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 312–323.

47. M. Izadi, A. Heydarnoori, and G. Gousios, "Topic recommendation for software repositories using multi-label classification algorithms," *Empiri-*

cal Software Engineering, vol. 26, 09 2021.

48. F. Herrera, F. Charte, A. J. Rivera, and M. J. del Jesus, *Multilabel Classification: Problem Analysis, Metrics and Techniques*, 1st ed.   Springer Publishing Company, Incorporated, 2016.

49. A. Blanco, A. Casillas, A. Pérez, and A. D. de Ilarraza, "Multi-label clinical document classification: Impact of label-density," *Expert Systems with Applications*, vol. 138, p. 112835, 2019.

50. F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Mlsmote: approaching imbalanced multilabel learning through synthetic instance generation," *Knowledge-Based Systems*, vol. 89, pp. 385–397, 2015.

51. M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

52. "Fast bert repository." [Online]. Available: https://github.com/utterworks/fast-bert

53. "Transformers documentation." [Online]. Available: https://huggingface.co/docs/transformers/index

54. Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training bert in 76 minutes," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=Syx4wnEtvH

55. G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," *Data mining and knowledge discovery handbook*, pp. 667–685, 2009.

56. R. B. Pereira, A. Plastino, B. Zadrozny, and L. H. Merschmann, "Correlation analysis of performance measures for multi-label classification," *Information Processing & Management*, vol. 54, no. 3, pp. 359–369, 2018.

57. M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

58. J. Romano, J. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys?" in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–3.

59. C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 683–711, 2019.

60. D. Petkovic, M. Sosnick-Pérez, K. Okada, R. Todtenhoefer, S. Huang, N. Miglani, and A. Vigil, "Using the random forest classifier to assess and predict student learning of software engineering teamwork," in *2016 IEEE Frontiers in Education Conference (FIE)*.   IEEE, 2016, pp. 1–7.

61. E. Goel, E. Abhilasha, E. Goel, and E. Abhilasha, "Random forest: A review," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 7, no. 1, 2017.

62. T. Pushphavathi, V. Suma, and V. Ramaswamy, "A novel method for software defect prediction: hybrid of fcm and random forest," in *2014 International Conference on Electronics and Communication Systems (ICECS)*. IEEE, 2014, pp. 1–5.

63. S. M. Satapathy, B. P. Acharya, and S. K. Rath, "Early stage software effort estimation using random forest technique based on use case points," *IET Software*, vol. 10, no. 1, pp. 10–17, 2016.

64. M. Van Gompel and A. Van Den Bosch, "Efficient n-gram, skipgram and flexgram modelling with colibri core," *Journal of Open Research Software*, vol. 4, no. 1, 2016.

65. T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PloS one*, vol. 10, no. 3, p. e0118432, 2015.

66. P. A. Flach and M. Kull, "Precision-recall-gain curves: Pr analysis done right." in *NIPS*, vol. 15, 2015.

67. A. Strauss and J. Corbin, *Basics of qualitative research techniques.* Thousand oaks, CA: Sage publications, 1998.

68. I. S. Wiese, R. Ré, I. Steinmacher, R. T. Kuroda, G. A. Oliva, C. Treude, and M. A. Gerosa, "Using contextual information to predict co-changes," *Journal of Systems and Software*, vol. 128, pp. 220–235, 2017.

69. R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, 1993, pp. 207–216.

70. D. Qiu, B. Li, and H. Leung, "Understanding the API usage in Java," *Information and software technology*, vol. 73, pp. 81–100, 2016.

71. C.-W. Seah, I. W. Tsang, and Y.-S. Ong, "Transfer ordinal label learning," *IEEE transactions on neural networks and learning systems*, vol. 24, no. 11, pp. 1863–1876, 2013.

72. G. Pinto, I. Steinmacher, and M. A. Gerosa, "More common than you think: An in-depth study of casual contributors," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*, 2016, pp. 112–123.

73. S. Balali, I. Steinmacher, U. Annamalai, A. Sarma, and M. A. Gerosa, "Newcomers' barriers. . . is that all? an analysis of mentors' and newcomers' barriers in OSS projects," *Comput. Supported Coop. Work*, vol. 27, no. 3–6, p. 679–714, Dec. 2018.

74. A. Sarma, M. A. Gerosa, I. Steinmacher, and R. Leano, "Training the future workforce through task curation in an OSS ecosystem," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 932–935.

75. G. H. L. Pinto, F. Figueira Filho, I. Steinmacher, and M. A. Gerosa, "Training software engineers using open-source software: the professors' perspective," in *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 2017, pp. 117–121.

76. M. Ferreira Moreno, W. H. Sousa Dos Santos, R. Costa Mesquita Santos, and R. Fontoura De Gusmao Cerqueira, "Supporting knowledge creation through has: The hyperknowledge annotation system," in *2018 IEEE International Symposium on Multimedia (ISM)*, 2018, pp. 239–246.

77. Y. Lu, G. Li, Z. Zhao, L. Wen, and Z. Jin, "Learning to infer API mappings from API documents," in *International Conference on Knowledge Science, Engineering and Management*.   Springer, 2017, pp. 237–248.

A Appendix

Additional data from RQ2 results. Some data were presented with box plots in section 5.2. The redundant data (and more detailed) about the experiments are available here in tables.

Table 17 Overall performance from models created to evaluate the corpus. Hla*

Model	Corpus	Precision	Recall	F-measure	Hla
TD-IDF	Title (T)	0.830	0.794	0.809	0.117
TD-IDF	Body (B)	0.840	0.786	0.811	0.116
TD-IDF	T, B	0.839	0.799	0.817	0.113
TD-IDF	T, B, Comments	0.831	0.796	0.812	0.116
BERT	Title (T)	0.616	0.592	0.596	0.277
BERT	Body (B)	0.599	0.598	0.591	0.27
BERT	T, B	0.595	0.559	0.568	0.269
BERT	T, B, Comments	0.597	0.587	0.582	0.266

* *Hla - Hamming Loss*

Table 18 Overall performance from models created to evaluate the number of grams. Hla*

Model	Precision	Recall	F-measure	Hla*
unigrams (1,1)	0.841	0.829	0.834	0.115
bigrams (2,2)	0.844	0.809	0.825	0.119
trigrams (3,3)	0.841	0.809	0.822	0.123
quadrigrams (4,4)	0.845	0.798	0.819	0.125

* *Hla - Hamming Loss*

Table 19 Overall performance from models created to evaluate the algorithms. Hla*

Model	Hla	Precision	Recall	F-measure
DecisionTree	0.105	0.861	0.837	0.847
Dummy	0.202	0.749	0.658	0.698
LogisticRegression	0.120	0.858	0.792	0.822
MLPClassifier	0.107	0.853	0.846	0.848
MLkNN	0.126	0.837	0.801	0.816
RandomForest	0.107	0.864	0.836	0.849
BERT	0.277	0.601	0.574	0.578

* *Hla - Hamming Loss*

We also include the confusion matrix for all projects trained and tested alone (Tables 21–26). The confusion matrix for the RTTS project is in Table 14 on Section 7.

Table 20 Overall performance from models created using the dataset with all projects merged to evaluate the algorithms

Model	Hla*	Precision	Recall	F-measure
DecisionTree	0.157	0.768	0.576	0.654
LogisticRegression	0.167	0.779	0.504	0.611
MLPClassifier	0.154	0.766	0.595	0.666
MLkNN	0.179	0.709	0.555	0.617
RandomForest	0.153	0.785	0.573	0.659
BERT	0.219	0.725	0.511	0.593

* Hla - Hamming Loss

Table 21 Overall performance from the selected model - JabRef project

API-domain	TN	FP	FN	TP	Precision	Recall
Network	13	6	8	19	0.76	0.70
DB	43	1	0	2	0.67	1
Interpreter	12	9	5	20	0.69	0.8
Logging	0	7	0	39	0.85	1
Data Structure	45	0	0	1	1	1
i18n	40	0	5	1	1	0.17
Setup	33	3	1	9	0.75	0.9
Microservices	42	0	4	0	0	0
Test	41	0	3	2	1	0.4
IO	0	6	0	40	0.87	1
UI	4	2	0	40	0.95	1
App	41	1	2	2	0.67	1

Table 22 Overall performance from the selected model - Powertoys project

API-domain	TN	FP	FN	TP	Precision	Recall
APM	72	3	11	7	0.70	0.39
Interpreter	91	0	1	1	1	0.50
Logging	92	1	0	0	0	0
Data Structure	90	1	0	2	0.67	1
i18n	92	0	1	0	0	0
Setup	47	6	11	29	0.83	0.72
Logic	87	1	0	5	0.83	1
Microservices	70	1	15	7	0.88	0.32
Test	91	1	0	1	0.50	1
Search	41	6	8	38	0.86	0.83
UI	25	15	1	52	0.78	0.98
Parser	87	1	2	3	0.75	0.60
App	22	12	0	59	0.83	1

Table 23 Overall performance from the selected model - Audacity project

API-domain	TN	FP	FN	TP	Precision	Recall
Util	38	3	2	13	0.81	0.87
APM	50	1	2	3	0.75	0.60
Network	54	0	0	2	1.00	1.00
DB	49	1	2	4	0.80	0.67
Error Handling	37	3	2	14	0.82	0.88
Logging	52	0	1	3	1.00	0.75
Thread	46	1	0	9	0.90	1.00
Lang	54	0	0	2	1.00	1.00
Data Structure	15	8	2	31	0.79	0.94
i18n	48	2	0	6	0.75	1.00
Setup	13	5	2	36	0.88	0.95
Logic	3	0	0	53	1.00	1.00
IO	10	3	6	37	0.93	0.86
UI	6	2	0	48	0.96	1.00
Parser	51	1	0	4	0.80	1.00
Event Handling	28	6	1	21	0.78	0.95
App	29	4	4	19	0.83	0.83
GIS	50	1	2	3	0.75	0.60
Multimedia	15	4	5	32	0.89	0.86
CG	50	0	1	5	1.00	0.83

Table 24 Overall performance from the selected model - MTT project

API-domain	TN	FP	FN	TP	Precision	Recall
NLP	34	0	9	9	1.00	0.50
APM	9	0	0	43	1.00	1.00
DB	37	0	4	11	1.00	0.73
Lang	10	1	2	39	0.97	0.95
DevOps	0	5	0	47	0.90	1.00
Setup	18	6	8	20	0.77	0.71
IO	43	0	4	5	1.00	0.56
UI	0	2	0	50	0.96	1.00
Security	9	4	2	37	0.90	0.95

Table 25 Confusion matrix and performance: Project JabRef - transfer learning.

API-domain	TN	FP	FN	TP	Precision	Recall
Network	65	7	43	3	0.30	0.07
DB	107	8	3	0	0	0
Interpreter	71	0	47	0	0	0
Logging	45	3	63	7	0.70	0.10
Data Structure	98	16	4	0	0	0
i18n	103	0	15	0	0	0
Setup	43	72	0	3	0.04	1
Microservices	77	24	12	5	0.17	0.29
Test	77	13	23	5	0.28	0.18
IO	33	0	67	18	1.00	0.21
UI	2	35	13	68	0.66	0.84
App	9	90	0	19	0.17	1

Table 26 Confusion matrix and performance: Project Audacity - transfer learning.

API-domain	TN	FP	FN	TP	Precision	Recall
APM	127	1	9	0	0	0
Network	132	1	4	0	0	0
DB	117	0	20	0	0	0
Error Handling	105	0	32	0	0	0
Logging	88	41	7	1	0.02	0.13
Thread	122	0	15	0	0	0
Lang	133	0	4	0	0	0
Data Structure	37	0	100	0	0	0
i18n	118	0	19	0	0	0
Setup	28	1	94	14	0.93	0.13
Logic	5	10	60	62	0.86	0.51
IO	31	8	65	33	0.80	0.34
UI	0	18	9	110	0.86	0.92
Parser	126	1	10	0	0	0
Event Handling	68	0	69	0	0	0
App	58	33	12	34	0.51	0.74