Towards Analyzing Contributions from Software Repositories to Optimize Issue Assignment

Vasileios Matsoukas, Themistoklis Diamantopoulos, Michail D. Papamichail and Andreas L. Symeonidis
Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki
Thessaloniki, Greece
vmatsouk@ece.auth.gr, thdiaman@issel.ee.auth.gr, mpapamic@issel.ee.auth.gr, asymeon@eng.auth.gr

***Abstract*—Most software teams nowadays host their projects online and monitor software development in the form of issues/tasks. This process entails communicating through comments and reporting progress through commits and closing issues. In this context, assigning new issues, tasks or bugs to the most suitable contributor largely improves efficiency. Thus, several automated issue assignment approaches have been proposed, which however have major limitations. Most systems focus only on assigning bugs using textual data, are limited to projects explicitly using bug tracking systems, and may require manually tuning parameters per project. In this work, we build an automated issue assignment system for GitHub, taking into account the commits and issues of the repository under analysis. Our system aggregates feature probabilities using a neural network that adapts to each project, thus not requiring manual parameter tuning. Upon evaluating our methodology, we conclude that it can be efficient for automated issue assignment.**

***Index Terms*—automated issue assignment, GitHub issues, issue triaging**

I. Introduction

Software teams nowadays host their code in online repositories and collaborate using issue tracking systems in order to implement features, fix bugs, plan releases, etc. This new paradigm has brought forth an abundance of data that contain valuable information about software projects, including not only source code (commits), but also information concerning project planning and monitoring (issues/bugs), and developer productivity. We argue that these data can be mined for improving the software development process.

Apart from documenting and orchestrating the resolution of bugs, the issues tracked by online repositories may signify and prioritize feature requests, keep track of development tasks or even organize release cycles. When a new issue is created in such as system, a member of the team (known also as *triager*) assigns it to a team member (or generally a contributor of the project). The process of optimally assigning tasks to team members is far from trivial, as it requires good knowledge of the project as well as the team, taking into account the past experience and the current status of each team member individually. As a result, task assignment may be time-consuming and even hard, considering that the triager may not have clear knowledge of every detail of the project, and, even when he/she does, he/she may still have to spend considerable time and effort to do the optimal assignment.

To solve this problem, several approaches for automating issue assignment have been proposed. Most of these approaches use data from bug tracking systems, such as Bugzilla[1], and employ various techniques to recommend the assignment of new issues/bugs according to past assignment data [1]–[6]. Other systems focus on building contributor profiles based on existing contributions and subsequently using them to find the most suitable engineer for the new task at hand [7]–[10]. The data and the techniques used vary in each case; some are based on text data and machine learning models [1], [2], while others may also incorporate information from source code [7]–[9], or use more sophisticated algorithms including semi-supervised models [3] or even deep learning [5], [6].

Although the aforementioned solutions are effective under certain scenarios, they also have important limitations. First of all, these approaches focus mainly on bugs, therefore they have not been evaluated on the broader scope of task/feature assignments. Secondly, they are largely based on properly using a dedicated bug tracking system, and as a result they may not be applicable in projects where the software team does not follow a strict process. In the same context, these approaches are often applied on a per-project basis and thus they may require manual parameter tuning. Finally, certain contemporary approaches consider only information related to bug tracking, without taking into account the EngOps/DevOps aspects of software development (e.g. commits, mentions) that could be useful for task assignment.

In this paper, we build a task assignment recommender that confronts the above limitations. Our system employs a dataset extracted from GitHub[2], including issues from a diverse set of repositories with varying number of contributors. Using GitHub issues favors the applicability of our system in cases where a more strict dedicated bug tracking system is not available. Apart from issues and issue comments, our system further takes into account the commits of each project and employs a model that aggregates all possible information to provide an assignment probability for each contributor of the project. Our model, which is based on a support vector machines classifier and a neural network, is fully adaptable to the project under analysis, producing efficient recommendations without manual parameter tuning.

The rest of this paper is organized as follows. Section II reviews the related work in the area of automated task

[1]https://www.bugzilla.org/
[2]https://github.com/

assignment and further probes on the limitations of existing approaches. Our methodology for building a new task assignment recommender is presented in Section III. Section IV evaluates our approach and illustrates its application on a software project. Any threats to the validity are discussed in Section V. Finally, Section VI concludes this work and provides interesting insight for future research.

II. Related Work

As already noted, the continuously increasing need for building better software, while remaining on-schedule and on-budget, has driven the research community towards constructing methodologies able to optimize the software development process. In this context and given the constant turn towards collaborative software development, as reflected in the existence of numerous code hosting facilities that involve millions of projects and collaborators, several research efforts are directed towards harnessing contributions' information for optimizing the software development process. These research efforts target various different outcomes, such as determining the workload required to resolve bugs [11], [12], identifying development characteristics for optimizing team building [13] or even performing more effective prioritization of requirements [14].

Among the aforementioned directions and given its vital importance towards effective project management, the challenge of determining the most appropriate developer to undertake a certain task (which is also widely known as "issue, task or bug triaging") has drawn a lot of attention. To that end, the majority of the proposed methodologies employs data originating from bug tracking systems in order to train classification models capable of identifying the developer that best fits to a certain task based on historical data regarding previous assignments. One of the first approaches towards automated bug triaging is that of Murphy and Cubranic [1], who employ data from bug reports that originate from the development of Eclipse project. According to their methodology, each bug report corresponds to a certain document containing a specific vocabulary and is assigned to a certain class of the problem, which refers to the corresponding developer. Using Naive Bayes for training their classifier based on the historical data, the achieved accuracy is around 30%.

Building on top of the aforementioned approach, Anvik et al. [2] identify the need to incorporate additional information (such as the current workload or the vacation schedule) into the bug assignment procedure in order to strengthen the effectiveness of the constructed models. Upon selecting Support Vector Machines for training their classifiers, they were able to improve the accuracy of the bug assignments. However, their approach does not result in hard assignments, but acts as a semi-automated assignment recommender providing a list with the top relevant developers where the triager is responsible to make the final assignments. In addition, given that information regarding labels and assignees is not always available for historical data, which results in significant information loss, Xuan et al. [3] employ semi-supervised learning using Expectation Maximization in order to incorporate into their dataset bug reports without labels/assignment information. Additional approaches that employ machine learning for automated bug assignments involve multi-label classification [4] or even deep learning using Recurrent and Convolutional Neural Networks [5], [6].

Other than focusing on machine learning, there are also approaches that perform assignments based on information regarding the individual skills and the expertise of developers, as identified by their contributions to the source code itself. Such approaches analyze code modifications and bug occurrences using natural language processing techniques in order to extract the expertise of the candidate developers as reflected in the dominant concepts of the code they modify and thus match them with the ones of the upcoming task [7]–[9]. Finally, there are also approaches that employ the DevOps principles and make use of the roles of individual developers as a contributing factor in order to assign to them the available tasks [10].

Another interesting direction in the area of automated bug triaging involves the analysis of the cases when a certain bug is re-assigned to a different developer due to the fact that it was not resolved in the first place, a procedure known as "bug tossing". To that end, there are approaches that harness information originating from tossing graphs and employ Markov Chain Models in order to recommend the most suitable developer using goal-oriented paths [15]–[17].

The approaches discussed in the previous paragraphs are effective in certain use cases, however they also exhibit several inherent limitations. Most of the approaches analyzed depend on bug tracking systems and focus mainly on assigning bugs based on textual information from past assignments [1], [2]. As a result, their use on broader scenarios, including e.g. the use of different issue tracking systems or the existence of issues beyond the narrow scope of bugs, is limited. Furthermore, certain systems may require manual parameter tuning to be applied on different projects, further impairing their applicability, while others do not take into account the commits of the project [4]–[6]. Finally, topic modeling approaches, which focus mainly on the problem of developer profiling [7]–[9], and bug tossing approaches [15]–[17], though interesting, deviate from the scope of this work.

In this work we propose a system that is applicable on the broader task/issue assignment scenario. Our system extracts information from GitHub commits, issues, and issue comments, and processes it to generate a set of features that are aggregated and modeled to provide efficient task/issue assignments. As the data may be sparse (and there may also exist missing data), we demonstrate how useful recommendations can be provided even when a software team employs a less rigorous issue tracking system, like GitHub. Furthermore, our system is applicable on a large set of diverse projects, as it exhibits automated parameter tuning to adapt to the specifics of each software project.

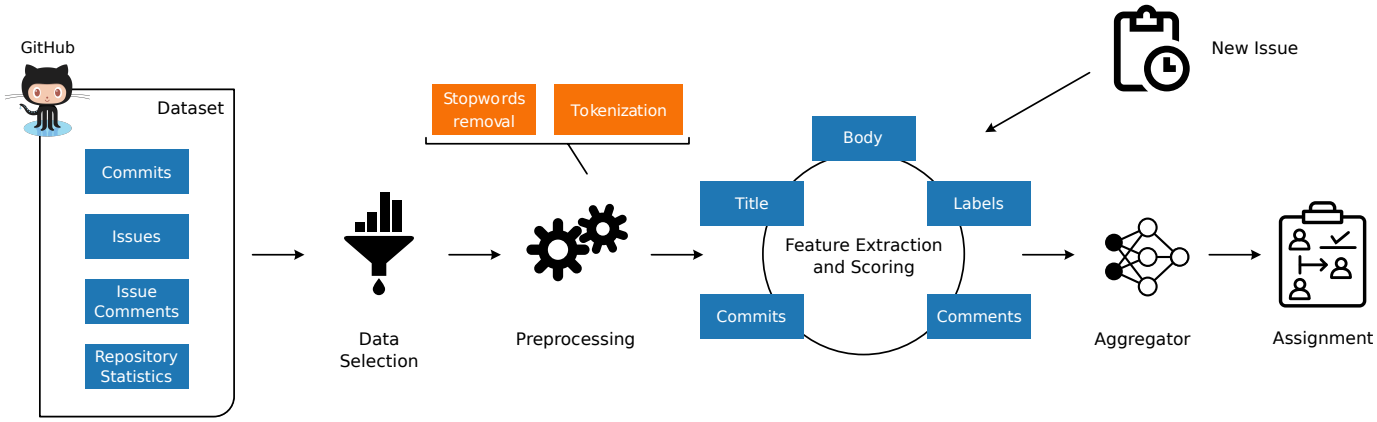

Fig. 1. Overview of the System Architecture

III. Methodology

In this section we present our methodology for automated issue assignment.

A. Overview

The architecture of our system is shown in Figure 1. The main functionality is summarized in four steps, which involve data extraction, data preprocessing, feature extraction and scoring/recommendation.

The data used for our analysis are extracted from a database of contribution data [18], which includes the individual contributions of more than 60,000 contributors involved in 3,000 repositories of GitHub. The data reside in a MongoDB instance and are organized in collections, which include information on issues, issue comments, commits, repository statistics, developer activity metrics, etc. Indicatively, the database contains more than 800,000 issues, 1,800,000 comments and 3,900,000 commits.

Our choice of this dataset is supported by the fact that it is diverse and covers a wide range of scenarios. As the dataset relies on GitHub, we focus on issues that are raised from contributors in software teams (i.e. used to track features or bugs) as well as external contributors (e.g. developers using an API). Furthermore, the repositories of the dataset rank among the most popular ones, as indicated by their star count, and therefore bear a high level of acceptance as well as respect the principles of modern software development. As they also vary in size and complexity, they cover a wide range of scenarios, allowing our approach to be rather domain-agnostic.

As a first step, we create a set of filtering rules according to which we determine the repositories and developers contributing to our analysis. In addition, we apply data labeling as well as text preprocessing techniques to remove unwanted textual content. Having constructed a clean and reliable dataset, we move on to build features in order to model the contributions of developers. The features employed are issue titles, bodies, labels as well as developers' comments and commits. Finally, we combine those features using two models, a simple average model and a neural network model. Thus, our methodology can be applied on repositories to determine the suitability of each contributor in terms of undertaking a certain issue. The components outlined in Figure 1 are presented in detail in the following subsections.

B. Data Selection

The first step entails the selection of appropriate repositories in order to build the ground truth for the forthcoming analysis. The filters applied to create our dataset, as well as the labeling method, are discussed in this subsection.

At first, upon investigation, we chose to filter out repositories with less than 50 and more than 5000 issues. The lower threshold ensures that the data are enough for effectively applying data mining techniques, while the upper one is chosen to cut out immense repositories. Moreover, we dismiss repositories with only 1 contributor, since in this case issue assignment is trivial. We further refine our selection by demanding the number of issues to be at least 10 times larger than the contributors of the repository. This choice guarantees a representative number of issues to conduct our analysis. Finally, we require that developers have a minimum support of 15 previously closed issues, so that there is a sufficient background to model their contribution.

The next step of dataset construction involves selecting the class attribute of an issue, which essentially means that we try to identify the developer who actually resolved it. Here, there are two fields that we can make use of, namely the 'assignee' and 'closed_by' fields. Consequently, we have to consider which field is superior and what happens in case of missing values. The answer to the first question, lies on the working principles of GitHub. The 'assignee' field is set by the author of the issue, implying that he/she considers a contributor as the most suitable to undertake that issue. Therefore, this field can be considered a reliable source for labeling an issue. On the other side, the 'closed_by' field indicates which contributor has closed the issue. However, given the specifics of the development process followed in each project, the person who closed the issue is not necessarily the one who really worked for its resolution (i.e. the team

leader only closes issues given that closing an issue refers to the acceptance of the respective functionality), thus we should be cautious when using it as ground truth for labeling the issue. The second challenge concerns missing values on those fields. It is possible that the author omits to set an 'assignee', or that an already closed issue lacks the name of the developer who closed it. What we propose as a labeling method is based on a three tier decision mechanism:

- If the 'assignee' field bears a valid username, then it is chosen as class attribute, as we deem such an explicit choice as highly reliable.
- If the former does not happen, then we examine the 'closed_by' field. We consider this field reliable, when the respective contributor does not focus on the operations part of DevOps [13]. Operations' engineers might confirm the delivery of tasks, and thus close issues without really being involved in their resolution. To find out whether this may be the case, we check if the commits made by the contributor at hand surpass the issues closed by him/her. If the commits are more than the closed issues, then we deem the contributor as dev-oriented and thus use the 'closed_by' field as a class attribute. Otherwise, the label is considered invalid and the issue is discarded.
- If both fields are empty, then the issue is discarded from the analysis.

C. Data Preprocessing

The main components of an issue are its title and body. Those components are the two main features used for assessing developers' involvement. Before, however, applying any text mining techniques, it is important to preprocess the data in order to filter out any noise and extract any useful information. To do so, we employ the NLTK [19] library and perform a series of preprocessing steps. At first, we remove any html tags and perform tokenization to remove any punctuation. After that, we discard numbers, single characters as well as stopwords found in the English stopwords list of NLTK. Finally, we convert all terms to lowercase and perform lemmatization. An example result of applying our text preprocessing mechanism is shown in Table I. It is evident, that after the preprocessing stage, the text content emphasizes better on the issues' key terms.

TABLE I
TEXT PREPROCESSING EXAMPLE

Input	Output
Posted this <a href-https://github.com/jediwhale/fistsharp/issues/79> issue on the fitsharp github site. If the output table of a fixture contains more columns than the input, Fitnesse throws an exception when the number of rows in the two tables is equal.	post issue fitsharp github site output table fixture contains columns input fitnesse throw exception number row two table equal

D. Feature Extraction and Scoring

In order to measure a contributor's suitability for undertaking an issue, we evaluate his/her contribution based on a set of extracted features. The purpose is to examine how closely is a contributor connected to a new issue from a semantic point of view. In our work, these features can be divided in two logical categories. The first refers to issue-oriented features and comprises components extracted from issue reports. This way we evaluate how a new issue relates to the issues that the contributor resolves. The issue-oriented features employed are the title, the body, and the labels of the issue. The second category focuses on contributor-oriented features that are derived from tracking developer activities. In this aspect, we attempt to discover the contributor's interest to certain types of issues. The contributor-oriented features consist of the contributor's comments and commits. The rest of this subsection describes how these features are extracted and used in ranking developers.

1) Issue Text: Issue text components, namely title and body, provide meaningful insight regarding the issues' nature as well as to the technical field they belong to. For example, an issue about GUI, screens, aesthetics etc. is probably well-suited to a front-end developer, whereas an issue about databases, SQL, nodes would better match a back-end developer. The purpose behind using issue text features is to quantify the connection between the problem description and the contributors' technical area.

The quantification requires that the text data are transformed to a representation that allows similarity comparisons. To do so, we employ a Vector Space Model, where each term is a dimension of the model and each document (title or body) is a vector of the model. We create two models, one for the issue titles and one for the issue bodies. The weight of each term in a vector is determined using *term frequency-inverse document frequency (tf-idf)*. Thus, for our collection of documents D, the weight of a term t in a document d is computed as follows:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{1}$$

where $tf(t, d)$ is the term frequency (tf) of term t in document d, while $idf(t, D)$ is the inverse document frequency (idf) of term t in the collection of documents D. The term frequency is computed as the square root of the number of times the term appears in the document, while the inverse document frequency is used to model how common are the terms in all the documents of the collection. The inverse document frequency of a term t in a document collection D is computed as follows:

$$idf(t, D) = 1 + log\frac{1 + |D|}{1 + |D_t|} \tag{2}$$

where $|D_t|$ is the number of documents containing the term t. Using this factor, very common terms, which may not be useful for training our model, are penalized.

The vectorized representation of issue titles and bodies is used to train a classifier, which will then be able to categorize new issues to a contributor of the repository. For this

multiclass classification problem, we used Support Vector Machines (SVM), and specifically the 'One-Vs-Rest' approach. Concerning the parameters of the SVM, since repositories may have quite diverse data, a set of parameters that may perform well in one repository may underperform in another. As a result, the parameters are selected for each repository individually using a parameter grid. The grid adds flexibility as to the choice of the kernel, the choice of the regularization constant C, the gamma factor and the degree of the algorithm (not all parameters apply to all kernels). The grid values of the parameters are shown In Table II, where the value 'scale' for gamma is 1 divided by the product of the number of features with the number of data samples.

TABLE II
SVM GRID PARAMETERS

Kernel	C	Gamma	Degree
Linear	$[0.1, 1, 10, 100]$	–	–
Polynomial	$[0.1, 1, 10, 100]$	–	$[2, 3, 4]$
RBF	$[0.1, 1, 10, 100]$	$[`scale', 0.001, 0.0001]$	-

The optimal configuration is chosen through grid search with 5-fold cross validation, optimizing the F-measure, computed for all issues. Upon training the SVM model, we compute the assignment probability of the issue under analysis to each contributor in the repository. This probability is calculated as the distance from the separating hyperplane and forms the contributor's score in title and body features.

2) Issue Labels: Attaching labels to issues implies the author's intention to classify the issue in a specific category. This action can be mined to investigate whether there is a pattern connecting the labels with the issues that a developer resolves.

For each repository under analysis, we initially extract all its labels. For each label, we examine the training issues and keep track of the contributor and the number of issues closed with that label. We apply this procedure for every label, and create a global index for the repository. When a new issue arises, the contributor receives a score which is the relative frequency of his/her involvement in that label. In case an issue has multiple labels, we first sum the frequencies of the contributor in each label, and then apply the previous step. The reasoning behind the labels scoring strategy lies in the assumption that developers tend to get involved to issues with certain labels.

To better illustrate our methodology on labels, we provide a practical example. Consider a repository with 3 contributors, "Contributor 1", "Contributor 2" and "Contributor 3", and 4 labels, namely "interface", "server", "storage", "http". We keep track of the involvement of each contributor in each label as in Table III. As shown in this table, the first contributor mostly handles "storage" issues, the second contributor tends to "interface" and "server" issues, while the third contributor focuses more on issues labeled with "interface" and "http". In a scenario where a new issue has the labels "server" and "http", we compute the scores by summing up the label frequencies for each contributor (e.g. $3 + 1$ for the first contributor, etc.) and dividing by the total occurrences of the two labels (which are equal to 31). The final scores will be $4/31$, $13/31$, and $14/31$ for contributors 1, 2, and 3 respectively.

TABLE III
EXAMPLE OF CONTRIBUTORS WITH ISSUES LABELS

Label	Contributor 1	Contributor 2	Contributor 3
interface	5	10	15
server	3	9	4
storage	12	3	1
http	1	4	10

3) Issue Comments: GitHub allows users to initiate a discussion on an issue by adding comments to it. A contributor's participation to the issue commenting thread typically implies his/her contribution to its resolution. It is also frequently the case that the resolver of the issue belongs to the commenters list. Given this observation, the contributors' comments can be a useful feature for issue assignment. However, when a new issue arises, the only known fields are its title, body and labels. Consequently, we can only make use of contributors' past commented issues and attempt to find out their connection with the new issue. Therefore, to evaluate the significance of issue comments in our methodology, we attempt to measure the connection between the past issue titles that developers comment on and the title of the new issue.

For each contributor, we retrieve the issues on which the contributor has commented in a 365-day time window since the appearance of the new issue. We impose this limit to mitigate the possibility of disorienting the model with developers that may have retired or change specialization field. Then, we employ the tf-idf model built for titles in subsection III-D1 to calculate the cosine similarity of the titles of the commented issues with the title of the new issue. The cosine similarity is defined for two issue titles/documents d_1 and d_2 as:

$$cossim(d_1, d_2) = \frac{d_1 \cdot d_2}{|d_1| \cdot |d_2|} = \frac{\sum_1^N w_{t_i,d_1} \cdot w_{t_i,d_2}}{\sum_1^N w_{t_i,d_1}^2 \cdot \sum_1^N w_{t_i,d_2}^2} \quad (3)$$

where w_{t_i,d_j} is the tf-idf score of term t_i in document d_j (computed using (1)) and N is the total number of terms. Finally, upon computing the similarities/values of all past issue titles with the title of the new issue, we compute their mean value. The mean is applied to issues with 10% larger value than the smallest observed, so that it is not distorted by the small or zero cosine similarity values. This value represents the score of the contributor based on the comments feature.

A practical example is given in the Table IV, which utilizes an issue from the repository "syncthing_syncthing-android", titled (after preprocessing) "syncthing binary crashed error code" and assigned to contributor 'Zillode'.

The repository has a team of 4 contributors. For the issue under analysis, we mainly focus on the title and the ground truth resolver. For each contributor, we present a list with their extracted past-commented issues. As it is observed, 'catfriend' and 'wweich' have not commented on any issues in the defined

TABLE IV
EXAMPLE OF CONTRIBUTORS WITH ISSUE COMMENTS

Issue ID	Title	Score
audris		**0.0913**
87	syncthing work sd card	0.0913
catfriend		**0.0000**
-	-	-
Zillode		**0.4709**
164	syncthing binary crashed error code ok	1.0000
282	use different default port syncthing gui	0.0964
288	still us syncthing	0.3165
wweich		**0.0000**
-	-	-

TABLE V
EXAMPLE OF CONTRIBUTORS WITH COMMITS

Commit ID	Message	Score
audris		**0.0830**
103	Support new UR value (fixes #471)	0.0830
catfriend		**0.0000**
-	-	-
Zillode		**0.8425**
153	Restart now when importing config (fixes #386)	0.8425
140	Limit retries when create the config file	0.2973
32	Updated to Syncthing v0.11.2	0.4616
wweich		**0.0000**
210	Updated Phone, 7 inch tablet and Android TV screen shots	0.3135
206	Update phone screenshots + new 7 inch tablet screenshots (ref #422)	0.2168

time window, thus receiving a zero score. Contributor 'audris' has commented on only 1 issue with a low but non zero similarity score. 'Zillode' has commented in 3 issues, getting a higher mean similarity score, as the titles of the relevant commented issues resemble closely the testing issue title.

4) Contributor Commits: Commits may prove quite useful for predicting the assignment of issues, since the aim of commits by contributors is issue resolution. Similarly to comments, the process for the commits takes into account the past commits of the contributor and compares them with new issues.

Thus, initially, for each contributor we retrieve the most recent commits. Once again, the retrieval concerns commits submitted in a 365-day window until the new issue, to capture the activity of contributors up to the latest time. After that, we compute the cosine similarity between the commit message of those commits and the title of the new issue. We use a vector space model similar to that of subsection III-D1, and, instead of the average, we take into account the maximum value. We use the maximum since when a contributor has even one commit message similar to an issue comment, he/she probably has worked in a relevant part of the source code. So, the fact that the contributor may have worked also in other parts should not influence the assignment.

An example is given in the Table V, where an issue retrieved from the repository "syncthing_syncthing-android", titled (after preprocessing) "restart config updated fix Zillode" is assigned to contributor 'Zillode'.

For each contributor, we extract their commits in the relevant time window. In Table V, for the contributors that have many commits, only the 3 most relevant are shown (in terms of maximum similarity score). As one may notice, 'catfriend' has no commits in the past time window, thus getting a zero score. Moreover, 'audris' and 'wweich' only have 1 and 2 commits respectively, which are weakly related with the issue in question. 'Zillode', on the other hand, has multiple commits, one of which with a commit message strongly related to the title of the issue in question, therefore getting the highest score in the commits section.

E. Recommending Issue Assignments

As already mentioned, the objective of our system is to recommend issue assignments. The final assignment recommendation is based on all five features discussed in the previous section, namely those based on the title, body, labels, comments and commits. Taking into account all available data, we initially create an aggregation approach based on averaging all scores for every contributor. In this case, the recommended assignee is the one who has the highest score.

Up to this point, we have built a model that considers versatile sources of information to perform issue assignment. However, it is not guaranteed that averaging over all five features can lead to the best possible model. In fact, each team may embrace a different way of building software, thus leading to a unique feature combination/weighting. Therefore, we propose a scheme that can cope with the challenges of various repositories. In practice, we attempt to build a model capable of adjusting the importance of those features according to each repository. We refrain from rule-based techniques, as these may result in certain thresholds that are not always adaptable and thus do not conform to different repositories. Instead, we build an adaptable model that provides output that is specific to the characteristics of each repository.

We employ a multilayer perceptron with one hidden layer. Our neural network receives as input the contributor's scores and calculates their influence on the output, which is a score in the range $[0, 1]$. Designing an efficient neural network is a challenging task. The main obstacle to overcome is the ability of the network to generalize efficiently not only in one, but in a number of repositories. To address this challenge, in our work, we propose a reconfigurable network architecture, where a set of the network parameters is dynamically adjusted to meet the needs of each repository. We try to keep the variable set as small as possible, to reduce computational complexity. Therefore the network consists of some constant and some variable/configurable parameters, which are shown in Tables VI and VII, respectively (parameters refer to the Keras API[3]).

[3]https://github.com/fchollet/keras

TABLE VI
CONSTANT NEURAL NETWORK PARAMETERS

Parameter	Value
Input, Output nodes	[5, 1]
Hidden Layers, Nodes	[1, 10]
Kernel Initializer,	
Bias Initializer	TruncatedNormal(mean=0.0, stddev=0.05)
Optimizer	Adamax(lr=0.001, beta1=0.9, beta2=0.999)
Loss Function	Binary Crossentropy
Activation Function	Sigmoid
Early Stopping	monitor='val_loss',min_delta=1e-2, patience=2,restore_best_weights=True

TABLE VII
CONFIGURABLE NEURAL NETWORK PARAMETERS

Parameter	Value
L2 Regularization	[0.00001, 0.0001, 0.001, 0.01]
Batch Size	[1, 4] · number_of_devs
Train Epochs	Automatically defined by Early Stopping

Upon experimentation, the constant parameters of Table VI have been proven to have stable performance among repositories. In contrast, we detected that three parameters, namely the L2 regularization, the batch size, and train epochs significantly affect the network performance. Specifically, the L2 regularization imposes a penalty to network weights, therefore playing an important role to the generalization ability. The batch size determines how many samples are received by the network before the update of its weights. We found that some repositories are favored by a slower rate, while others require a faster one. Last but not least, the train epochs are also a configurable parameter. However, this parameter is automatically tuned by an early stopping mechanism which monitors the validation loss function. If there is no decrease at least by 0.01 for two consecutive epochs, then the network terminates the training process. The early stopping technique helps avoid overfitting on the training data, while allowing an extra configurable parameter without computational load. All in all, the design choices of the configurable parameters are shown in Table VII. The optimal configuration is chosen through exhaustive grid search with a 5-fold cross validation method, optimizing the F-measure, computed for all issues.

IV. EVALUATION

In this section, we evaluate our methodology for automated issue assignment. We focus on the effect of adding information to the model from the different feature sources analyzed. Additionally, we examine the effectiveness of the neural network optimized model on top of all features. For this purpose, we assess 5 different configurations, starting from a baseline model with only issue text features, gradually incorporating labels, comments, and commits, and finally the neural network layer. In short, the configurations examined are:

- Title & body
- Title & body & labels
- Title & body & labels & comments
- Title & body & labels & commits
- Optimized model

The training stage of our models is based on an incremental learning technique, which is commonly used for assessing bug triaging systems [9], [10]. In practice, the testing set is split in ordered folds of the same size, and the system makes predictions on the issues of the current fold, considering all the previous issues. In the next round, the current fold is incorporated in the training set, and we make predictions for the next fold etc. In this way, the system is able to continuously update its knowledge on recent issues. For the purposes of incremental learning, the issues are sorted in chronological order based on their 'created-at' attribute. In our work, we begin with 70% of the issues as training set, while the test fold length is set to 5%. Consequently, the system goes through 6 rounds of training and after each round, the training set is extended by the testing fold. Finally, we impose the restriction that the training set has at least one sample from each class in the initial round, so that the prediction is meaningful. If this requirement is not met, the system skips this training round and moves on to the next fold.

As issue assignment is in fact a multiclass classification problem, the number of contributors in a repository is a crucial factor to our model's performance. It is generally expected that larger teams will impose a higher level of difficulty to our problem compared to smaller ones. In Figure 2, we present the distribution of contributors in the analyzed repositories. As it is observed, the repositories ranging from 2 to 5 contributors make up the 53% of total repositories. For 6 to 15 contributors, we also gather an acceptable number of repositories, although the samples are more limited as it gets harder for such a number of contributors to meet our filtering criteria.

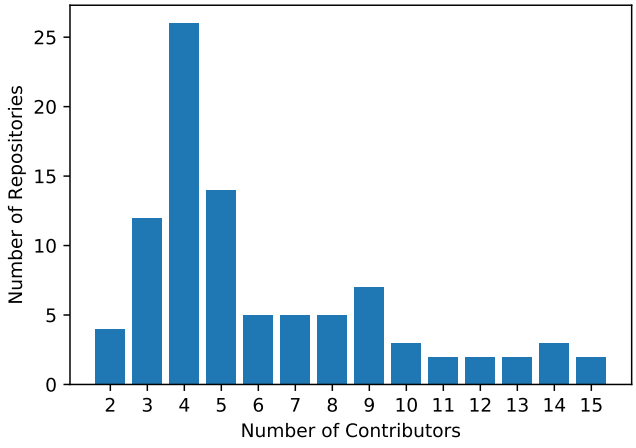

Fig. 2. Number of contributors per repository

As a result, we focus our evaluation on three axes. First, we study the accuracy of our methods for repositories with different numbers of contributors. Next, we evaluate our approach against the different baselines outlined above, using the main classification metrics (precision, recall, accuracy, and

F-measure). Finally, we present an example analysis on a repository to better illustrate the effectiveness of our approach in a qualitative manner.

A. Contributor Analysis

The first part of our evaluation focuses on the impact of the different number of contributors on the performance of our models. Specifically, Figure 3 captures the mean accuracy achieved by the various configurations, depending on the number of contributors in the repository. As one may notice, using only title and body features, we achieve relatively poor predictions in the range of 2 to 4 contributors. By gradually incorporating all the features, the model exhibits a boost in performance achieving accuracy values close to 0.8 and 0.7, for 2 and 3/4 contributors respectively. The application of the neural network, with the optimal feature scheme, further increases the effectiveness of the system. In the range from 5 to 10 developers, we observe that the combination of all features maintains a stable accuracy response close to 0.55, while the optimized model achieves slightly better performance.

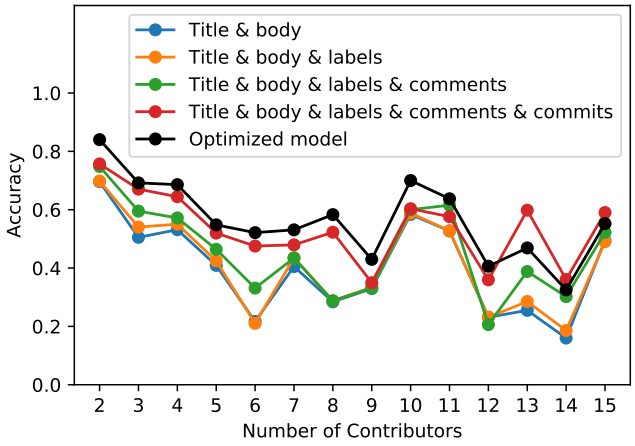

Fig. 3. Accuracy versus number of contributors per repository

Finally, in the range from 11 to 15 developers, the performance begins to show a reasonable descending trend as the model has to predict many classes. Some outbreaks are observed in this zone, as a result of the few repository samples that make this analysis sensitive to outliers. By the term "outlier", we refer to repositories that exhibit significant differences in the way they utilize issues information against the expected behavior. E.g. a repository where contributors use comments instead of assignees to perform assignments exhibits a falsified behavior, which is reflected in the analyzed features and thus is expected to achieve poor performance. Although using only title and body leads to low accuracy scores, the full-feature and optimized models maintain a decent performance in that range. And of course, even with a baseline combination of features, the results surpass those of a random classifier for the corresponding number of developers. For instance, a random-based model would yield assignment accuracy equal to 0.2 for a team of 5 contributors, whereas our model is on average almost three times more accurate.

B. Classification Evaluation

In the second part of our evaluation, we assess the different configurations in terms of accuracy, precision, recall and F-measure. Figure 4 depicts the performance of each configuration on the aforementioned classification metrics, averaged over all repositories. At first glance, the results confirm that incorporating features from different sources leads to a more robust model, and that the optimal feature scheme further boosts the performance of our system.

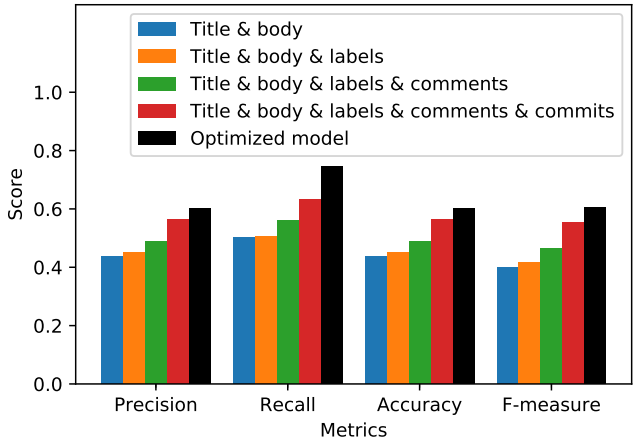

Fig. 4. Evaluation metrics for all configurations, averaged over all repositories

In the context of the task assignment challenge, accuracy can be seen as the global percentage of successful assignments, and precision as the percentage of the suggested developers who actually worked on the issue. In specific, these two metrics appear to follow the same trend, with our best configuration achieving a value of 0.6. However, the most remarkable aspect of our system is its performance regarding the recall metric. In practice, our system on average achieves recall values close to 0.8. Considering that recall is perceived as the percentage of developers who worked on the issue and were actually recommended, our system can become a valuable guide in recommending issue assignments. Last but not least, the F-measure, which is the harmonic mean of precision and recall, further confirms our assumptions about the effectiveness of the various configurations.

C. Repository Analysis

To conclude our evaluation, we conduct a qualitative analysis to confirm that our system can prove useful in practice. For the purpose of this case study, we have chosen the repository 'airlift/airlift' as it exhibits typical project characteristics. The project initiated in 2010, has more than 700 (both open and closed) issues by the time the database was updated, and has a core of 5 major contributors that satisfy our filtering criteria. In fact, the repository has a total of 37 contributors, but the

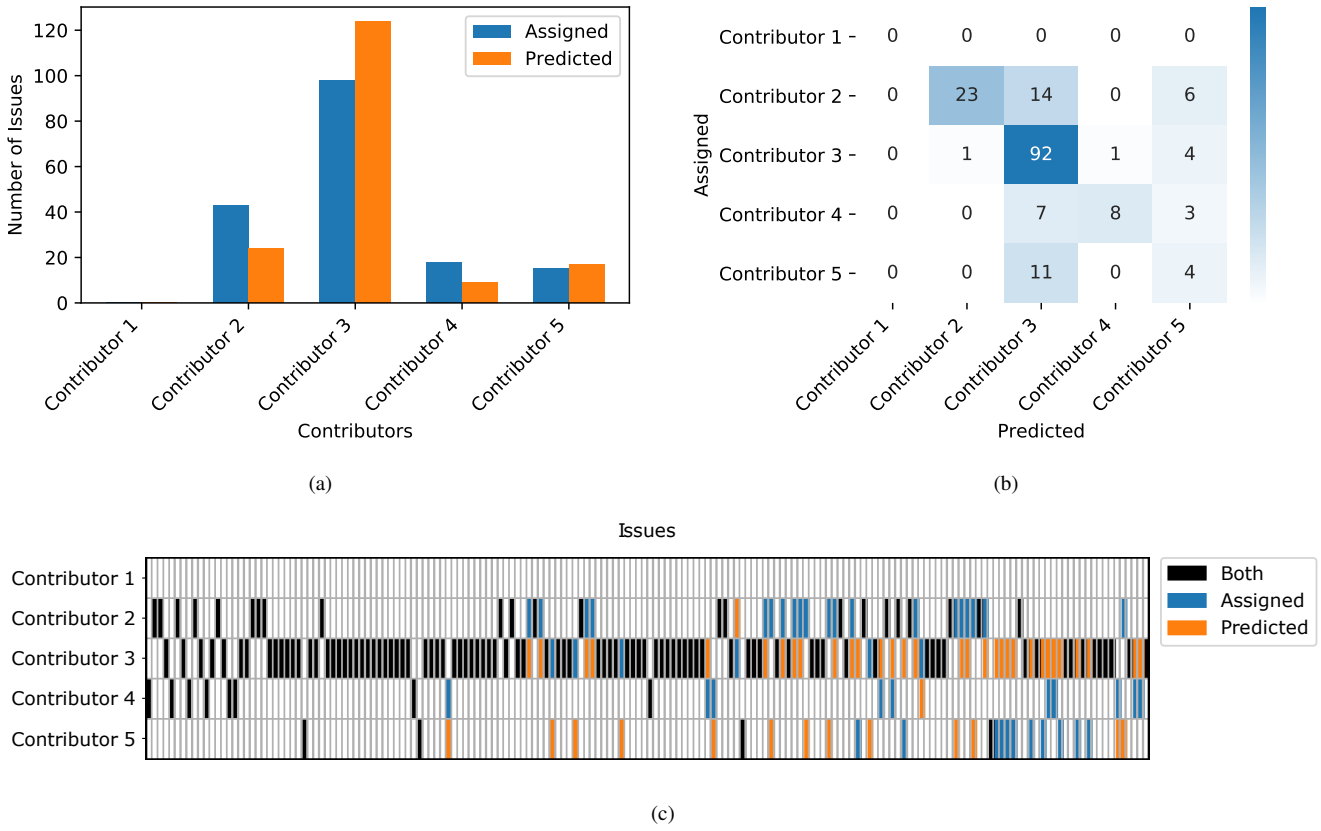

Fig. 5. Repository analysis, which includes (a) the number of issues assigned and predicted per contributor, (b) the confusion matrix of our model, and (c) the distribution of issues (assigned and predicted) on all contributors of the repository

contribution of other than those 5 members is negligible, and thus reasonably set aside by our model.

As already mentioned, our model tests on the latest 30% of total issues. This corresponds to approximately 210 test issues in our case, which eventually fall down to 174 after eliminating open and untracked issues. The number of issues assigned and predicted per contributor is shown in Figure 5a. At first glance, we may note that our system manages to efficiently capture the workload capacity among the team members. Specifically, the system maintains the balance, assigning more issues to prominent members, while not abusing those with less regular activity. The equivalent confusion matrix for those assignments is given in Figure 5b. As shown in that matrix, our model correctly recommends most of the assignments as the majority of observations fall in the main diagonal. There are also some misclassified issues, which however should not necessarily be treated as unreasonable assignments. In fact, although several issues that should be assigned to contributors 2, 4, or 5 are assigned to contributor 3 by our model, these issues could be resolved by either of these contributors in terms of knowledge and skills.

Figure 5c depicts the distribution of issues (assigned and predicted) to all contributors, in the timeline of the project. It is evident that contributor 3 is the most involved contributor, and undertakes the majority of issues. Clearly, our model detects the skills of this contributor and effectively shifts the balance towards him/her. At the same time, the model neither neglects nor overloads contributors 2, 4 and 5, which have less, yet regular contribution. Therefore, our model exhibits its capability to cope with imbalanced classes. This is further confirmed by inspecting the first third of the timeline where the assignments are highly accurate, despite some misclassifications during the later progress of the project. As already argued, however, these are issues that could have equally been resolved by either of the developers. Lastly, contributor 1 is neither assigned nor recommended any issues throughout that period, which verifies that the model has acted correctly in that case as well.

On top of the above analysis and in an effort to assess whether the assignment results are reasonable with respect to the expertise of each contributor, we analyze information from individual contributions in order to extract the primary directions each member of the software team. In this context, we apply tf-idf in order to identify the primary directions of each contributor based on the issues he/she is involved into. This identification includes the creation of a corpus that contains all concepts included in the project as keywords. We perform the same analysis for each contributor using the title and the body of all issues he/she is involved into and we calculate the frequencies of the co-occurring concepts. Table VIII presents the results for all five contributors of the project

analyzed in this subsection, including the top 5 keywords for each contributor.

TABLE VIII
PRIMARY CONTRIBUTIONS DIRECTIONS

Contributor	Dominant Keywords
Contributor 1	—
Contributor 2	jetty, server, add, thread, request
Contributor 3	test, error, fix, update, java
Contributor 4	add, http, request, jvm, java
Contributor 5	log, http, support, client, add

Given the results of this table, contributors 2, 4, and 5 appear to primarily undertake the implementation of new features, as reflected by the dominant keywords of their contributions. Upon manually examining the commits of the respective repository, we found out that the majority of their contributions involves additions of new source code rather than modifications and deletions (the percentage of additions is around 70% of the contributions). On the other hand, contributor 3 appears to be responsible for testing, as the majority of his/her contributions refer to fixing errors that occur upon testing (e.g. the titles of the issues assigned to this contributor include "fix launcher for Python $<$ 2.7", "Fix sporadic failure in testHttpRequestEvent", etc.). As before, this is also reflected in the difference between the added and removed lines of code. For this contributor, the deleted lines of code are twice as many as the added ones (32,093 versus 67,199 at the time of writing). All in all, the aforementioned patterns are also reflected in the assignments recommended from our strategy and thus verify that our models are able to capture the contribution characteristics of the analyzed projects.

V. THREATS TO VALIDITY

In this section, we discuss any threats to the validity of our approach as well as the steps taken to mitigate them. At first, concerning our selection of repositories, this was based on their popularity as defined by the number of Github stars and the number of contributors, which may not be representative for projects that follow specialized development principles. However, given that these repositories cover a diverse set of different scopes, we may safely assume that our methodology performs efficiently on the majority of software development projects, from a domain-agnostic point of view. Should one require the analysis of more domain-specific repositories, the training set should be modified appropriately to incorporate the custome features of such projects.

Concerning missing data, we may note that the quality of information provided is a factor that affects the performance of our issue assignment recommender. In specific, there are cases where the issue title and body fields are omitted, or the issues are left unlabeled or even mislabeled. Additionally, the number of commits and comments may be comparatively small to make reliable assignments. However, this is a challenge faced by several approaches in this field, as issue management systems may sometimes include incomplete or missing information on projects. To mitigate this problem, our approach includes a preprocessing/filtering step in order to use high quality data for the construction of the models.

Finally, comparison with existing approaches was not directly feasible, given that most approaches use different datasets (and not GitHub) and thus they are optimized under a different scope. Traditional issue management systems are typically used by software teams for specific purposes (e.g. to drive development or track bugs), while GitHub issues are somewhat generic, as they may be used not only within a software team for development, but also as a communication channel with users/developers external to the team. We plan, however, as future work to assess the applicability of our methodology on other issue management systems, and evaluate it with respect to the current state-of-the-practice.

VI. CONCLUSION

As more and more software teams employ issue tracking systems to keep track of the progress of their projects, automated issue assignment has grown to be an important research area for software development. In this work, we have designed a recommendation system that can effectively assign GitHub issues to different contributors in a repository. Our system takes into account repository data offered by GitHub, i.e. commits, issues, and issue comments, and builds a feature set that can be used to produce probabilities for different assignments of an issue. Another major point of our system is the fact that it employs an updatable model that can adapt its parameters to the specifics of different projects/repositories.

Our evaluation indicates that our system is indeed quite robust, producing better results than the baselines in diverse repositories with different number of contributors, and as a whole. Furthermore, upon illustrating the application of our system on a sample repository, we may conclude that it produces reasonable recommendations, and thus can be used effectively for automated issue assignment.

Future work lies in several directions. At first, we could extract more types of data, including e.g. the source code of the commits for each contributor, the workload of the team with respect to the issue, etc. Another idea would be to further investigate the histories of issues to identify patterns (e.g. multiple re-assignments) that affect the performance of our model and/or build a reliability index in order to produce more accurate assignments. Furthermore, we could further investigate our choice of models, especially by employing text embeddings to improve the text mining component of our methodology. An interesting research direction would also be to apply our methodology on other issue tracking systems, beyond GitHub, and assess its performance on these systems. Finally, an interesting idea for future research would be to conduct a survey in order to assess how our system can be used in different scenarios.

REFERENCES

[1] G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. Citeseer, 2004.

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.

[3] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," *arXiv preprint arXiv:1704.04769*, 2017.

[4] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 72–81.

[5] S. Mani, A. Sankaran, and R. Aralikatte, "Deeptriage: Exploring the effectiveness of deep learning for bug triaging," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, 2019, pp. 171–179.

[6] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong, "Applying deep learning based automatic bug triager to industrial projects," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 926–931.

[7] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *2009 6th IEEE international working conference on mining software repositories*. IEEE, 2009, pp. 131–140.

[8] F. Servant and J. A. Jones, "Whosefault: automatic developer-to-fault assignment through fault localization," in *2012 34th International conference on software engineering (ICSE)*. IEEE, 2012, pp. 36–46.

[9] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 2–11.

[10] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 22–30.

[11] J. Cabot, J. L. C. Izquierdo, V. Cosentino, and B. Rolandi, "Exploring the use of labels to categorize issues in open-source software projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 550–554.

[12] S. Akbarinasaji, B. Caglayan, and A. Bener, "Predicting bug-fixing time: A replication study using an open source software project," *journal of Systems and Software*, vol. 136, pp. 173–186, 2018.

[13] M. D. Papamichail, T. Diamantopoulos, V. Matsoukas, C. Athanasiadis, and A. L. Symeonidis, "Towards extracting the role and behavior of contributors in open-source projects," in *14th International Conference on Software Technologies (ICSOFT)*, 2019.

[14] M. Alkandari and A. Al-Shammeri, "Enhancing the process of requirements prioritization in agile software development-a proposed model." *JSW*, vol. 12, no. 6, pp. 439–453, 2017.

[15] L. Chen, X. Wang, and C. Liu, "Improving bug assignment with bug tossing graphs and bug similarities," in *2010 International Conference on Biomedical Engineering and Computer Science*. IEEE, 2010, pp. 1–5.

[16] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–10.

[17] S.-Q. Xi, Y. Yao, X.-S. Xiao, F. Xu, and J. Lv, "Bug triaging based on tossing sequence modeling," *Journal of Computer Science and Technology*, vol. 34, no. 5, pp. 942–956, 2019.

[18] T. Diamantopoulos, M. D. Papamichail, T. Karanikiotis, K. C. Chatzidimitriou, and A. L. Symeonidis, "Employing Contribution and Quality Metrics for Quantifying the Software Development Process," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR 2020)*. New York, NY, USA: ACM, 2020.

[19] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, Inc., 2009.