A Developer Recommendation Method Based on Code Quality

Matheus Camilo da Silva
Graduate Program in Informatics
Pontificia Universidade Catolica do Parana
Curitiba, Brazil
matheus.camilo@ppgia.pucpr.br

André Armstrong Janino Cizotto
Machine Learning Committee
Siemens Ltd
Curitiba, Brazil
andre.janino@siemens.com

Emerson Cabrera Paraiso
Graduate Program in Informatics
Pontificia Universidade Catolica do Parana
Curitiba, Brazil
paraiso@ppgia.pucpr.br

***Abstract*—During the development cycle of a project, it is common for software requirements and functionality to change and for code errors to occur. To deal with these unforeseen changes, the artifact known as change request, which is a formal proposal to alter a system, is used. Its assignment is an important step in the development process. Projects can receive a very high number of requests daily, which makes the automation of this process compelling. This work proposes a method for assigning unresolved requests, based on developer's profiles. The proposed method consists of three steps. The first step is to extract code quality metrics, commit data and previously resolved requests, in order to model developers through the mining of repositories. The second step concerns with the selection of the profile of potential developers through the application of natural language processing and information retrieval techniques. And finally, in the third step the appropriate developers are selected based on the quality of their code and the impact of their commits. Results from experimental evaluation show that the method is able to recommend more developers with a positive impact on the repository quality if compared to the *iMacPro* method.**

***Index Terms*—Software quality, clustering, developer recommendation, developer profile**

I. Introduction

Collaborative software development is an activity carried out by participants with different experiences, expertise and behaviors, that play different roles in the development of a project [1]. Regardless of the development paradigm used, writing a concise and detailed project specification is essential to the success of a project. A project specification is an important artifact that assists the development process by providing a description of the functional and non-functional requirements of a project.

During the development cycle, a project's specification may change and collaborators must be selected to make such changes [2]. However, due to the dynamic nature of the collaborative development ecosystem, the task of finding the most appropriate developer to perform such changes can be a daunting effort. For effective maintenance and quality assurance, it is necessary to evaluate the characteristics of a developer that can represent their work [3]. This assessment is often carried out with subjective criteria that may not reflect the real skill of a developer or the environment in which they finds themselves in, as shown by the work of [4], who conducted a survey with several professionals working in the industry, mostly in management positions within development teams. The result of the aforementioned survey indicates that the assignment of developers to change requests tends to be carried out manually through subjective heuristics.

Notwithstanding, depending on the size of a project, recommending developers to address change requests may not be feasible due to a myriad of factors, such as time and resource constraints. Therefore, an automated approach should be considered. Most methods found in the literature address automated developer recommendation through one of the following premises: the developer that is best suited is the most experienced with similar requests [5] [6], or that the most appropriate developer can be selected not only through the analysis of past requests, but also from data mining of code repositories [7] [8] and/or code defects. The proposal presented in this paper follows the second approach. When taking into account only the level of experience of developers as a factor for the recommendation process, code quality is put at risk, as the impact code changes may have is not assessed. Furthermore, the side-effects of distinct work patterns of a given developer, such as quantity and quality of changes for a particular task, is not addressed by the [5] model.

Moreover, this work proposes a method for assigning unresolved requests based on developer profiles, which is structured in three steps. The first step extracts code quality metrics, commit data and previously resolved requests through repositories mining in order to model developers. The second step concerns with the selection of candidate profiles through the application of Natural Language Processing (NLP), Information Retrieval (IR) and Machine Learning (ML) techniques. Lastly, the appropriate developers are selected based on the quality of their code and the impact of their commits. With that, requests are recommended to developers who are best fit for a given task, maintaining the overall project's code quality.

The remainder of the paper is organized as follows: Sections 2 and 3 present background information on developer recommendation systems and related work, introducing key concepts to understand of the present paper. Section 4 describes the process of data acquisition. In Section 5, the proposed method is presented, which is evaluated in Section 6 and its results analyzed in Sections 7 and 8. Finally, Section 9 concludes this paper with a brief summary of the conducted research.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

II. Change Requests Management

Software development is becoming more complex and collaborative due to the increase in demand and complexity [9] of projects, leading to the involvement of developers with different work patterns and characteristics. Moreover, the use of source code management tools, such as version control systems (VCS) and defect management platforms, is essential to coordinate the processes of software development. These platforms allow collaborators to register tasks related to bug fixing and development of new features [10], that is, software artifacts also known as *change requests*.

In general, defect management tools lean towards a change request (issue) resolution cycle as presented in Figure 1. The cycle starts with the creation of a new issue, which is composed of textual elements that describe the requested change. After its creation, the triage process takes place, where it is assigned to one or more appropriate developers for its resolution. What characterizes a developer as the most "appropriate" varies in the literature and in the development ecosystems [4]. The selection of the right assignee is fundamental, as each alteration may affect quality aspects of the project's code, such as maintainability and testability [11].

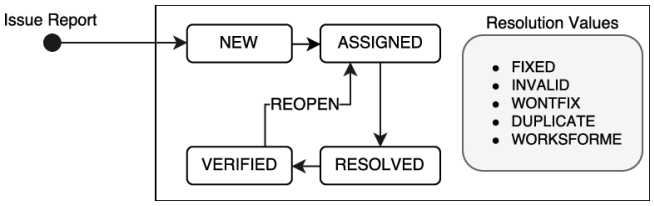

Fig. 1. The life cycle of a Change Request [12]

After the end of the triage step, the assigned developer analyzes the change and assigns a resolution value regarding the status of the issue. A *FIXED* resolution indicates that the change has been accepted and resolved, while *WONTFIX* means that the assignee is not address the issue and *INVALID* expresses that the change is not relevant. The value *DUPLICATE* points out that a very similar or identical issue already exists on the platform and *WORKSFORME* means that the problem described in the issue was not possible to reproduce, which may indicate that the problem is in the author's environment. From the creation of a resolution value, the issue is considered resolved and must be verified in the next step. An issue can be resolved and not necessarily generate code changes.

The last stage of the cycle involves the verification of the changes performed by the assignee, assessing the value of the generated resolution and the requested change. If the collaborators in charge of checking are not satisfied with the resolution value, the issue can be re-opened and assigned to another developer, starting the cycle again. If the assessment is positive, the issue is closed.

The whole process is recorded in the software management tool, which makes it a rich source of data to support the creation of developers' profiles based on code changes and interaction between developers in a collaborative environment.

III. Related Work

Although no work has been found in the literature that models developers according to the requirements proposed in this work, relevant related works that broaches the creation of developer profiles based on historical quantitative and qualitative features are presented in this chapter.

A. User Modeling

The methods presented by [13] and [14] characterize developers through a quantitative analysis of their work, in contrast to [15], which profile developers based on subjective perceptions of productivity. Despite seeking objective characteristics of the development activity, both methods are lacking, as only one aspect of the development process is evaluated, the quality of the developer code at [13] and the developer expertise at [14]. The former assumes that the quality of the code produced by a developer does not relate to the type of activity, and the latter considers that the expertise of a developer is unrelated to the quality of code produced.

B. Developer Recommendation

In the scope of this research, the recommendation model regard developers as "items" to be recommended resolve change requests. The literature presents several papers that propose different approaches to the task of assigning the most suitable developer for the resolution of bugs, the addition of new features and/or other change requests in general. Broadly speaking, the approaches for recommending developers have a similar structure, in which the main goal is to represent change request through different *NLP* techniques. This allows the identification of similar change requests based on the historical analysis of the repository, and with so, the recommendation of developers that carried out analogous change requests.

The paper [5] proposes an approach to predict severity and recommend developers to maintain software *bugs*. The proposal uses the *KNN* classifier and information retrieval techniques to search the historical reports of *bugs* and retrieve a set of reports semantically similar to the maintenance request of interest. The list of possible developers is ordered by the authors who have the highest number of maintenance performed most similar to the target maintenance request. Likewise, the severity of *bug* for this request is given by the severity of *bugs* for similar requests.

Moreover, the paper [16] opted to use a developer-based relationship network approach and expert assessment, consisting of a sequence of steps. The first step regards the recovering of maintenance requests for *bugs* that are more similar to the target request, using natural language processing and clustering techniques. From these similar requests, features such as authorship and maintenance time are extracted, which allows the construction of a network of developers' relationships. Candidate developers are selected based on the number of relationships in the network and through expert evaluation,

being sorted based on their experience and efficiency in the task of maintaining *bugs*.

Finally, no work has presented a method of recommending developer profiles for resolving change requests, based on code quality, expertise and context.

IV. Data Preparation

In order to create the developer model proposed by this paper, it is necessary to mine code repositories in version control systems in order to collect information about the change request screening process in a collaborative development environment, in addition to the quality of the code after the changes. The model requires data that expresses features of code quality, commits and textual elements of implemented change requests.

A few projects developed on the GitHub versioning control system were selected to be mined. The criteria for selecting projects were: number of contributors; volume of commits; number of repositories and programming language used. These criteria were used to establish parameters that would allow a massive data acquisition in different scenarios. The mining of code repositories was done in two open source projects: Elastic and Google. The Table I better details the repositories.

TABLE I
SELECTED REPOSITORIES FOR BASE CONSTRUCTION

Project	Repository	Developers	Commits	Closed Issues
Mockito	mockito	177	5,200	750
Google	guava	218	5,000	2,500
Google	closure-compiler	453	15,000	1,700

A. VCS's Data Extraction

GitHub has an error management tool called *Issues*, available in all projects developed on its platform [17]. An issue has certain textual elements with the purpose of elucidating characteristics of the alteration requested to the development team, such as title, description, comments of collaborators about the alteration and an id of the issue.

The GitHub platform allows the mining of source code repositories, commits and requests through its API [1], enabling the acquisition of the database for creating the developer model. The API is in its third version, it provides data in JSON format, and has a limit of 5000 requests per hour.

To carry out the data acquisition process of the repositories shown in Table I, it was necessary to create a tool to perform data collection through integration with the GitHub API. Given that it is in the interest of this work to collect only issues that have generated code changes, the tool collects all "closed" of each repository since the beginning of development. That is, data collection is limited to change requests that were assigned to a developer who performed code changes and completed the screening cycle described in Figure 1.

[1]https://developer.github.com/v3/repos/commits/

After obtaining all issues relevant to the method proposed by this paper through the GitHub events API, the acquisition tool extracts data for each repository from all commits that meet the aforementioned criteria. Notwithstanding, a single commit can resolve multiple issues, while multiple commits can address a single issue. In cases where one commit is related to several issues, the tool simply links each issue to it. In cases where there are multiple commits for the same issue, only the commit that is present in the main branch of the repository is taken into account, as only commits in that branch have code that is part of the official version of the repository.

Lastly, the tool collects data from the source code files that were changed by commits filtered in the previous step. Once the repository code is obtained, it is possible to extract quality metrics. While the quality of changes can be observed subjectively in the code, it is necessary to use tools that extract the code quality metrics for an objective assessment.

B. Extraction of Quality Metrics

As the repositories for this work were selected in order to obtain the largest possible number of viable collaborators, issues and commits, they tend to be bulky repositories, occupying a relatively large memory space. This makes the management of such repositories in several versions a costly task of time and resources, making it difficult to manually extract quality metrics for each commit. As the proposal of the work involves evaluating the quality of changes in a non-invasive way, it was necessary to employ an extraction tool capable of being applied in any repository without the need to be reconfigured at each extraction.

This present work used a tool called CK to classify the code quality metrics proposed by Chidamber and Kemerer [18] in repositories developed in Java without having to compile [19]. Originally, CK did not contemplate the extraction of metrics in different versions of the same repository, so the tool was adapted to enable this use case. Ultimately, the tool extracts a set of quality metrics per commit.

C. Acquired Data

The extraction of the data made available by the *VCS* from the selected repositories (Table II) and the quality metrics of their change requests generated the database used by this work. The database is presented in Table II and can be accessed on https://bit.ly/2GBcnhV.

TABLE II
SELECTED REPOSITORIES FOR BASE CONSTRUCTION

Repository	Issues	Commits	Files
mockito	184	286	1,797
guava	662	221	2,558
closure-compiler	1,035	548	4,394

V. Method

This paper proposes a method for recommending developers to resolve change requests in collaborative development environments. From the textual features of a new change request

in a repository, the proposed method is able to find a set of collaborators with the greatest capacity to handle the present task, and select from within that group the developer who will have the least negative impact on the health of the repository when implementing the required changes.

The method was designed according to the following assumptions:

- The tool implemented from the method must have access to a versioning platform with code repository, change request management and *commits*;
- The analyzed projects must strictly follow the object-oriented programming paradigm;
- The method must collect data from developers in a non-invasive way, so that it does not influence any aspect of development;
- The method will consider as a potential developer to make a change to a project, only those who have already resolved some change request previously;

The proposed method does not foresee human interference to carry out the assignment of change requests to developers. The main processes of the method can be essentially divided into three stages.

1) Based on machine learning techniques and human-computer interaction, representations of the characteristics of developers related to resolution of change requests are created. The product of this step is a base of developer profiles, where individuals in a group have similar characteristics.
2) The textual features of a target request are preprocessed and employed to find similar closed requests through the use of LSI, which is then used to select the authors of the suggested requests as candidate developers.
3) Candidate developers are ordered according to their impact on code quality. The developer with the least negative impact is recommended for the resolution of the target request.

A. User Modeling

The developer model proposed by this paper is a dynamic model, since the characteristics of a developer may change over time as more data is collected. According to [20] the task of building dynamic models is organized in the following phases: Model composition, Representation, Acquisition, Learning and Maintenance of the Model. Each phase in turn is characterized by the application of techniques and methods as described in the following subsections.

1) Model Composition: For the proposed dynamic developer model's composition, it is necessary to identify which data is relevant to the representation of behavioral aspects of developers in a collaborative development environment. After reviewing the literature and defining the availability of repositories, it was established the need to use three different types of historical data for the composition of a model that would assist in assigning developers to change requests: Data on the quality of the code of a developer for inferring how well he solves a request; Data about *commits*, which reveal information about the size of changes made by that developer; Data on the nature of the requests resolved by him / her, in order to establish the concepts that he / she has expertise.

2) Representation: As the dynamic model has a machine learning phase, the use of the *arff* format was established to represent the developer model proposed by this work. This file format describes a list of instances that share a set of features, and it is used in dynamic models because it has, like an Extensible Markup Language (XML), a flexible structure with the ability to easily add or remove elements [21].

The code quality features are described using different code metrics, chosen according to their influence on the objective assessment of software quality and the resources available by the CK extraction tool [19]. The metrics **WMC**, **DIT** and **NOC** are part of the set of metrics from Chimdaber and Kemerer [18], the latter being one of the most important sets of object-oriented metrics [22], and more widely referenced [23]. The metrics of Chimdaber and Kemerer were applied in many researches to monitor the quality of software under development [24] by establishing ranges of values for measurements of interests. The relevant metrics to the proposed developer model are:

- **CBO**: Coupling between objects
- **DIT**: Depth Inheritance Tree
- **NOSI**: Number of static invocations
- **RFC**: Response for a Class
- **WMC**: Weight Method Class
- **LCOM**: Lack of Cohesion of Methods
- **NBD**: Max nested blocks
- **LOC**: Lines of code

The data present in *commits* of repositories in version control systems is intended to characterize a developer's activity in the repository. This data provides information about the developer's behavior regarding factors such as frequency and size of the code change. The relevant data present in *commits* relevant to the proposed developer model are:

- Id repo
- Id author
- Id *commit*
- Creation date
- *Commit* additions
- *Commit* deletions
- *Commit* total changes

The features that identify the developer's expertise in relation to the areas of knowledge necessary for the resolution of requests for changes, can be found in the requests themselves. Each change request has textual elements that are created by the requesting author in order to describe the nature of the required change, such as: title, description and comments.

3) Acquisition: In this step, the method performs feature extraction from resolved historical requests, starting with the textual characteristics of closed issues. This is done by transforming all the textual elements of requests into documents, where each document represents the concatenation of the title and description of a request. A *corpus* is created from these

documents, which is indexed using the application of Latent Semantic Indexing. This technique creates a unique signature (index) for each document [25], as each index is a set of term values that represent the conceptual content of a document. Finally, the K-means machine learning technique is applied to the corpus, in order to group documents based on the similarity between their indexes. The group ID is saved by the method.

By extracting a resolved and assigned request, the extraction of the *commit* related to its resolution is initiated. The data about *commit* and its repository, is obtained through the Github API. From the extraction of the previous set, the repository files are obtained in the *commit's* version. The files are extracted through the *CK* tool, and quality metrics are calculated only for files changed by the developer in the *commit* related to the change request.

4) Machine Learning: Once the data that composes the developer's model has been acquired, it is possible to carry out the task of creating developer profiles. This task is responsible for creating different profiles based on the analysis of the dynamic model's instances of developers. This induction can be performed through both supervised and unsupervised learning techniques [20].

In the research carried out by [13], a series of Machine Learning algorithms were proposed for profile generation. Due to the relationship between the level of the programming course that the participants of the experiment were registered, with their performance during the performance of the same task, It was possible to use classifiers to create developers profiles in [13],

However, for the scope of this work, it is not possible to induce relations in an objective and automatic way for any developer in any project based only on repository characteristics. Due to the nature of these data and the proposal itself, it is feasible that profiles may be inferred by unsupervised learning techniques.

In order to induce developer profiles, the *k-means* clustering algorithm was used. The algorithm is able to divide the model's base population into groups of instances with respect to the similarity of its attributes. Thus, the groups are composed of change requests data that were solved in a similar way in the same repository. The ideal number of clusters for each repository was determined using the so-called "elbow method", which consists of executing k-means for a range of values, and calculating the sum of squared distances from each point to its assigned center (inertia) for each run. When plotted, the graph of inertia resembles a curved arm, where the "elbow" indicates the optimal value for *k* Figure 2.

5) Model Maintenance: Software development is a complex and dynamic task, which entails that the model proposed by this work must follow a dynamic approach. The model allows data exchanges, in order to track the progress of developers over time. There are two ways to perform model maintenance: the model is updated by explicitly/manually changing the profile; or the model is updated implicitly, where the data is obtained in a continuous and non-invasive way [20]. The maintenance of the proposed model can be carried out periodically as new requests are resolved by the developer recommendation of the proposed method.

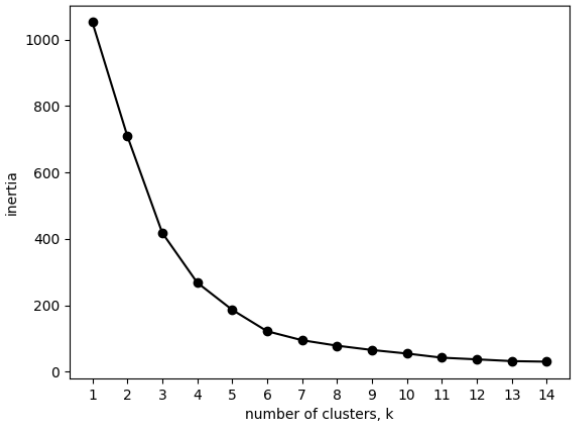

Fig. 2. Elbow method on the Guava repository

B. Candidate Developers

The first task of the method assignment step is preprocessing the target request. This task consists of applying Natural Language Processing techniques to the text found in requests' elements (title, description and comments). This task aims to eliminate textual noise in order to extract features that help to represent the context of the target request, facilitating the selection of similar historical requests through information retrieval techniques. After the application of NLP techniques for the preprocessing of the target requisition, the search for developers begins.

As an example, one can take into account the texts of a request presented below:

```
Currently deleting snapshots can be a
very slow process if the delete entails
removing a large number of blobs from
the repository (i.e. when one or more
indices that consistent of many files
becomes unreferenced and has to
be deleted).
```

After the application of tokenization, removal of *stop words* and stemming, the result is:

```
Currently deleting snapshot slow
process delete entail removing large
number blob repository.
one index consistent many file
becomes unreferenced deleted.
```

The texts of historical requests are indexed and added as documents in a *corpus* created in the first stage of the method. Latent Semantic Indexing generates a vector space where each document is represented by a sequence of values that indicate the relevance of topics to its textual content. The method uses cosine similarity in order to find in this vector space, the document (solved request) with the highest similarity index. From the found request, the proposed approach retrieves from the profile base, the list of candidate developers who have historically worked on requests belonging to the same group as the found request, as shown in the figure Figure 3.

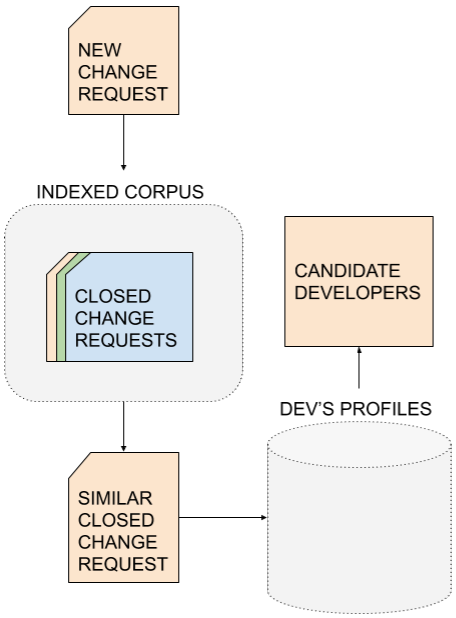

Fig. 3. Selection of Candidate Developers

C. Developer Recommendation

The last step of the method has as its main task to recommend the most appropriate developer for the target change request. Given the list of candidate developers for resolving the request, it is necessary to select the most appropriate contributor. Adhering to the premise that the most appropriate developer has the least number of bad quality metrics within his group, the list of candidates is ordered according to code quality metrics *commits*.

The sum of the normalized metric values for all *commits* in relation to the target request is calculated for each candidate developer.

VI. Results

There are currently no code bases that have performed their screening processes as proposed in this paper, which hardens the evaluation step as there is no gold model to be compared with. Therefore, evaluating the accuracy of the proposed method is not be sufficient, as developers of a test set repository may have been chosen by subjective characteristics, or even randomly [26].

The metric proposed in [1] was chosen to assess whether the developers recommended by the proposed method would have an equal or better impact on the code quality of a change compared to the actual author. This metric aims to measure the degree of developers contribution in collaborative environments. The metric can be described in the equation below where X is the value of a quality metric and i is the *commit* number:

$$\sum_{i=0}^{n} X_i - X_{i-1} \tag{1}$$

If the difference between sequential commits is high, it means that there is an increase in the value of the quality metric being evaluated and consequently a negative influence on the health of the repository and a poor degree of contribution. Therefore, the developers recommended by the proposed method must have a degree of contribution better or equal to the average of all their code quality metrics in relation to the developers who originally made the evaluated change.

However, it is also necessary to evaluate the accuracy of the proposed method in order to validate the profiles of the developers. Since the profile base is created from the history of requests resolved by collaborators in a repository, the developer who originally solved the evaluated issue must be at least on the list of candidate developers, even if he is not the most appropriate. To evaluate this part of the method, the metric *recall@k* was selected, as it is successfully used by similar works in the literature [27]. According to [26], *recall* is the proportion in which relevant items are found in the recommended items and its modification "*@k*" refers to the application of the metric for recommended item sets with different sizes.

The equation used to evaluate *recall* in different set sizes is shown below, where N_{rs}refers to the number of recommended developers who are the original authors of changes, and N_r, which refers to the number of recommended developers:

$$Recall@k = \frac{N_{rs}}{N_r} \tag{2}$$

The proposed approach was evaluated on 409 change requests from three different repositories that used the Java programming language, as presented in Table I. As the number of change requests is relatively small, an exhaustive cross-validation method as employed in order to use all of the data set's instances as part of the validation set, by the leaving-one-out method for each repository separately. The model was trained using all but one of the data set's instances and validated using the particular instance left out, repeating this process for each instance until all possible ways to divide a repository's data set into a training set and a validation set of one have been done. The method was evaluated using both the

Contribution (Equation 1) and *Recall@k* (Equation 2) metrics, as presented in the evaluation session.

In order to better evaluate the developer recommendation method proposed by this paper, its results must be compared to a state-of-the-art recommendation method. For this task, we selected the *iMacPro* developer recommendation method, as it requires only an access to a repository's source code and its change history. The *iMacPro* approach is akin to the one proposed by us with the most notable exception of using change proneness of a relevant unit of source code as a key factor for recommending developers instead of quality metrics. It uses *Latent Semantic Indexing* to locate source-code units that may relate to an incoming change request and rank them based on their historical change frequency. Finally, any developer that has contributed to those relevant units of source code are put together on a list forming the best-fit candidates for resolving the incoming change request. This list of developers is ranked based on the number of changes they made to the relevant units and the date in which they made them, favoring developers who worked more and recently [27]. The *iMacPro* approach recommends the top *n* developers of that list without creating a machine learning model.

Both methods were applied for each of the acquired repositories data, one repository at a time. Table III presents the recall values for the proposed method in contrast to the *iMacPro*. The recall value tends to increase as the number of *k* increases for both methods. The statistical *Student's t-test* was applied to determine if the means of the recall values produced by both methods are significantly different from each other. The performed *t-test's* resulted in a t-value of 0.30537 and p-value of 0.382012. As the obtained p-value is higher than 0.05, there is no statistically significant difference between the compared methods, which indicates that the proposed approach can recommend developers as well as a well established approach from the literature.

TABLE III
RECALL@K MEASUREMENT FOR THE PROPOSED METHOD

	Recall@k	Proposed Method	iMacPro
Mockito	**1**	0.11	0.13
	3	0.40	0.22
	5	0.81	0.29
Closure-Compiler	**1**	0.13	0.15
	3	0.28	0.48
	5	0.50	0.54
Guava	**1**	0.11	0.13
	3	0.22	0.28
	5	0.25	0.34

With the evaluation of proposed method's accuracy for recommending sets of developers to resolve change requests, the need to assess the impact on software quality that recommendations may have still remains. Table IV presents the values for the *contribution* metric described in [1] for both developer recommendation methods. In this evaluation, every developer in a repository had their *contribution* calculated, then the *contribution* of the most appropriate developer for resolving a change request, recommended by each method, was compared to the original change request's assignee. Moreover, the statistical *Student's t-test* was applied in a subset composed only of *Positive* values in order to determine if there is a significant difference between the recommendation made by the evaluated methods regarding software quality. The results obtained from the test leads to the conclusion that there is a significant difference at $p < 0.05$, with a t-value of 3.34783 and p-value of 0.014313.

TABLE IV
DEVELOPER CONTRIBUTION MEASUREMENT FOR THE PROPOSED METHOD

	Contribution	Proposed Method	iMacPro
Mockito	**Equal**	17	20
150	**Negative**	8	71
Change Requests	**Positive**	125	59
Closure-Compiler	**Equal**	25	28
188	**Negative**	3	96
Change Requests	**Positive**	160	64
Guava	**Equal**	12	15
118	**Negative**	15	62
Change Requests	**Positive**	91	41

Results from experimental evaluation show that 93% of the recommended developers have equal or higher levels of *contribution* than evaluated change request's assignees, in contrast to only 49% obtained by the *iMacPro* method.

Conclusively, since our method is based on the extraction and modeling of not only historical commit data but also code quality related to resolved change requests, in contrast to the *iMacPro* approach that uses change history as well but does not take into account the changes in code commit by commit, our method requires more computational effort to acquire data and to create the developers profiles. However the evaluation's results shows that there's a statistically difference between both methods concerning the recommending of developers with a better impact to a repository's code quality. It indicates that the trade off can be worth it when taking technical debt into account. Moreover once the acquisition step is done, the proposed method's maintenance step can be done without a high computational effort.

VII. CONCLUSION

In this work we proposed a method for assigning unresolved change requests based on developer profiles. Those profiles are build from clustering historical developing data.

The outcome of the evaluation phase demonstrates that the proposed approach can be beneficial to a repository's health, as it takes into consideration the quality of a developer's previous alterations when recommending a change request assignee. Moreover, while it has presented a similar *recall@k* result when compared to the *iMacPro* developer recommendation

method, it presents a superior performance in terms of assigning developers with a better track record in terms of code quality for the evaluated task.

To further validate the method a real world application is under development using real code from a company in the energy market. This would also allow us to explore possible recommendation models with features that cannot be found on a open code base, such as a developer schedule and position in the company.

ACKNOWLEDGMENT

We would like to thank Siemens Ltd for partially supporting this work financially.

REFERENCES

[1] P. R. BASSI, G. M. P. Wanderley, P. H. Banali, and E. C. Paraiso, "Measuring developers' contribution in source code using quality metrics," in *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*, May 2018, pp. 39–44.

[2] N. Ali and R. Lai, "A method of requirements change management for global software development," *Information and Software Technology*, vol. 70, pp. 49–67, 2016.

[3] N. E. Fenton and M. Neil, "Software metrics: roadmap," in *Proceedings of the Conference on the Future of Software Engineering*. ACM, 2000, pp. 357–370.

[4] y. Cavalcanti, I. Machado, P. Anselmo da Motal S. Neto, and E. Santana de Almeida, "Towards semi-automated assignment of software change requests," *Journal of Systems and Software*, vol. 115, 02 2016.

[5] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *J. Syst. Softw.*, vol. 117, no. C, pp. 166–184, Jul. 2016. [Online]. Available: https://doi.org/10.1016/j.jss.2016.02.034

[6] M. M. Rahman, G. Ruhe, and T. Zimmermann, "Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, Oct 2009, pp. 439–442.

[7] X. Sun, H. Yang, X. Xia, and B. Li, "Enhancing developer recommendation with supplementary information via mining historical commits," *J. Syst. Softw.*, vol. 134, no. C, pp. 355–368, Dec. 2017. [Online]. Available: https://doi.org/10.1016/j.jss.2017.09.021

[8] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, "Assigning change requests to software developers," *Journal of Software Maintenance*, vol. 24, pp. 3–33, 01 2012.

[9] M. Mohtashami, T. J. Marlowe, and C. S. Ku, "Metrics are needed for collaborative software development," *Journal of Systemics, Cybernetics, and Informatics*, vol. 9, no. 5, pp. 41–47, 2011.

[10] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*, May 2007, pp. 1–1.

[11] S. Lock and G. Kotonya, "An integrated, probabilistic framework for requirement change impact analysis," *Australasian Journal of Information Systems*, vol. 6, no. 2, 1999.

[12] M. Rakha, C.-P. Bezemer, and A. E. Hassan, "Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval," *Empirical Software Engineering*, 12 2017.

[13] F. Beal, P. R. de Bassi, and E. C. Paraiso, "Developer modelling using software quality metrics and machine learning," in *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems, Volume 1, Porto, Portugal, April 26-29, 2017*, S. Hammoudi, M. Smialek, O. Camp, and J. Filipe, Eds. SciTePress, 2017, pp. 424–432. [Online]. Available: https://doi.org/10.5220/0006327104240432

[14] E. Constantinou and G. M. Kapitsaki, "Identifying developers' expertise in social coding platforms," in *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2016, pp. 63–67.

[15] A. N. Meyer, T. Zimmermann, and T. Fritz, "Characterizing software developers by perceptions of productivity," in *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE Press, 2017, pp. 105–110.

[16] T. Zhang and B. Lee, "How to recommend appropriate developers for bug fixing?" 07 2012.

[17] GitHubInc. (2008) Mastering issues. [Online]. Available: https://guides.github.com/features/issues/

[18] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design," 1991.

[19] M. Aniche, *Java code metrics calculator (CK)*, 2015, available in https://github.com/mauricioaniche/ck/.

[20] I. Barth, "Modelando o perfil do usuário para a construção de sistemas de recomendação: um estudo teórico e estado da arte," *Revista de Sistemas de Informação da FSMA*, vol. 6, pp. 59–71, 2010.

[21] R. Robu and V. Stoicu-Tivadar, "Arff convertor tool for weka data mining software," in *2010 International Joint Conference on Computational Cybernetics and Technical Informatics*, May 2010, pp. 247–251.

[22] T. H. A. Soliman, A. El-Swesy, and S. H. Ahmed, "Utilizing ck metrics suite to uml models: A case study of microarray midas software," in *2010 The 7th International Conference on Informatics and Systems (INFOS)*. IEEE, 2010, pp. 1–6.

[23] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[24] R. Plosch, H. Gruber, C. Korner, and M. Saft, "A method for continuous code quality management using static analysis," in *2010 Seventh International Conference on the Quality of Information and Communications Technology*. IEEE, 2010, pp. 370–375.

[25] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.

[26] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.

[27] M. K. Hossen, H. Kagdi, and D. Poshyvanyk, "Amalgamating source code authors, maintainers, and change proneness to triage change requests," in *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 2014, pp. 130–141.