An Online Developer Profiling Tool Based on Analysis of GitLab Repositories

Jing Wang, Xiangxin Meng, Huimin Wang, and Hailong Sun(✉)

School of Computer Science and Engineering, Beihang University, Beijing, P.R. China
sunhl@buaa.edu.cn

Abstract. There are more and more developers as software is playing increasingly important roles in today's economic and social development. As a result, evaluating developers' expertise scientifically has become an urgent need for both Internet companies and developers. However, it seems that there is no satisfactory method to meet this demand currently. In this paper, we propose a solution to profile developers by analyzing their source code. We conduct the analysis of developers in terms of code quantity, code quality, skills, contribution, personalized commit time, and projects they participated in based on the GitLab code repositories. And we comprehensively evaluate developers' expertise from four perspectives, which are code quantity, code quality, contribution and score of projects they participated in. Compared with existing methods, our evaluation indicators are more comprehensive. We design and implement an online tool that can provide developer searching and profiling. Our tool has been used in Neusoft and Wonders Group to characterize the expertise and performance of their software developers.

Keywords: GitLab · Developer expertise · Developer profiling · Program analysis.

1 Introduction

The number of developers has increased year by year with the rise of the Internet industry. And the question how to evaluate and rank millions of developers has gotten more and more attention.

Many Internet companies are using GitLab systems to develop and manage projects. From study of Marlow et al. [1], we can know that due to the transparency of data and the difficulty in falsifying activity trajectories, the expertise of job seekers inferred from GitLab data is more reliable than the description on the resume. The activities and contribution history of developers can demonstrate their knowledge of the code [2–4]. Therefore, we can use these data to learn about the developers' expertise in software development. So we believe that we can gain a deep understanding of developers' expertise by analyzing the data on GitLab.

A comprehensive analysis and a reasonable evaluation of the developers' expertise based on GitLab code repositories are of great significance. In this paper,

we analyze GitLab developers from code quantity, code quality, skills, contribution, personalized commit time, and projects they participated in. And we use four indicators, which are code quantity, code quality, contribution and score of projects they participated in, to measure the developers' expertise. Moreover, we design and implement an online tool that can provide developer searching and profiling. Our tool can bring the following benefits: 1) Optimize task assignment. Team leaders can allocate relatively urgent or important projects to employees who can complete tasks with higher quality and efficiency. 2) Improve the employee evaluation mechanism. The management can conduct employee assessments based on the results our tool gives. 3) Enrich the developer selection method. Companies can judge job seekers' expertise according to their code and finally find excellent candidates. Our tool has been used in both our own laboratory and enterprises such as Neusoft and Wonders Group to characterize the software developers' expertise and performance.

The rest of the paper is organized as follows. Section 2 presents related work. In Section 3, we provide a detailed introduction of our proposed analysis method and evaluation indicators. Section 4 describes our use cases. We conclude this paper in Section 5.

2 Related Work

The research on GitHub is relatively mature. Since GitLab is very similar to GitHub, we learn some methods from GitHub's research and use them to analyze GitLab data. We mainly pay our attention to the following two aspects:

1.The developers' contribution. By analyzing contributing.md researchers found a method to judge how much every developer contributes to the project [5]. Gousios et al. [6] added the number of non-comment lines of code and the contribution factor function to evaluate the developer's contribution.

2.The developers' ability assessment. Li et al. [7] used fuzzy analytic hierarchy process to evaluate developer's capabilities from both social and technical attributes. Ke et al. [8] proposed a fuzzy DEA model, which used the information of code quantity, development time, number of commits, number of bugs, and leader's opinion, to evaluate the efficiency of developers. Constantinou et al. [9] quantified every developer's skills by using his commit activities and ranked the developers by programming languages.

3 Analysis of Developers on GitLab

In this section, we introduce our analysis of GitLab developers from code quantity, code quality, skills, contribution, personalized commit time, and projects they participated in.

In order to make different kinds of data have the same order of magnitude, the following equation is used to normalize data:

$$y' = \frac{y - y_{min}}{y_{max} - y_{min}} \tag{1}$$

where y is an element in the set, y_{min} is the minimum value in the set, and y_{max} is the maximum value in the set.

3.1 Analysis of Code Quantity

The most basic indicator for evaluating a developer's expertise is his code quantity, which visually shows his workload.

We use the number of non-comment lines of code written by the developer to measure his code quantity and define the indicator *dloc* to evaluate every developer's code quantity as

$$dloc = \frac{ncloc - ncloc_{min}}{ncloc_{max} - ncloc_{min}} \tag{2}$$

where *ncloc* is the number of the non-comment lines of code developed by the current developer, $ncloc_{min}$ and $ncloc_{max}$ are respectively the minimum and maximum value in the set of developers' number of non-comment code lines.

3.2 Analysis of Code Quality

Code quality is an important manifestation of developer's expertise. For newcomers, whose code quantity is small, the quality of the code can measure their expertise more reasonably. In addition, the quality rather than length of experience is a strong correlate of job performance after the first two years [10].

We use the SonarQube to analyze the code quality. SonarQube is an open source code quality assessment system that supports analysis of more than 20 programming languages. It is widely used in the industry [11, 12].

We evaluate the developer's code quality in terms of bugs(reliability), vulnerabilities(security), code smells(maintainability), complexity, and duplicated lines, and propose *dqua* as an indicator of the developer's code quality. We recount the number of bugs, vulnerabilities, and code smells so that each violation is counted up to once per file. We introduce the following five sub-indicators to measure the quality of every developer's code more scientifically:

$$k\_rate = \frac{16 * blo\_k + 8 * cri\_k + 4 * maj\_k + 2 * min\_k + inf\_k}{ncloc} \tag{3}$$

$$complexity\_rate = \frac{complexity}{ncloc} \tag{4}$$

$$duplicated\_lines\_density = \frac{duplicated\_lines}{lines} \tag{5}$$

Equation (3) calculates sub-indicators for evaluating bugs, vulnerabilities, and code smells, where k can be replaced by $bugs$, $vulnerabilities$, and $code\_smells$. The blo_k, cri_k, maj_k, min_k, inf_k respectively represent five levels of violations: blocker, critical, major, minor, and info. And *ncloc* represents the number of non-comment lines of code. Because different levels of violations have

different impacts on code, we give different weights to different levels of violations to measure the quality of the code better.

Equation (4) calculates the complexity rate of the code, where *complexity* is the overall complexity of the developer's code, and *ncloc* is the number of non-comment lines of code of the developer.

Equation (5) calculates the density of duplicated lines of the code. In the equation, *duplicated_lines* is the number of duplicated lines of the code developed by the developer, and *lines* is the number of lines developed by the developer.

Finally, we normalize the five sub-indicators with Equation (1) and evaluate the developer's code quality using the Equation (6).

$$dqua = \frac{\sum_k k\_rate' + complexity\_rate' + duplicated\_lines\_density'}{5} \quad (6)$$

3.3 Analysis of Skills

We demonstrate every developer's skills from both programming language distribution and keywords of programming fields. After understanding the skills of the developers, the team leaders can assign each member the tasks he is good at, which can help the tasks be completed more efficiently.

We keep a file collection for each developer. The programming language distribution of each developer is obtained according to the file extensions in each collection. And the programming languages that the developer is good at are known.

We treat the content of the files modified by the developer as an article, and use the LDA model [13] to analyze the programming fields that each developer is good at.

We connect a developer's changed lines in all the commits into a string as an article. Then we preprocess the data based on natural language processing, which includes eliminating noise, word segmentation, and deleting the stopwords in the text. We choose the LdaMallet model, which can automatically find the optimal values of α and β. The value of iterations is 700. By comparing some LdaMallet models with different number of topics, we make K be 3.

The trained model's topics and their keywords are shown in Table 1. It is obvious that the topic 1 is a combination of front and back ends, the topic 2 is related to the back end, and the topic 3 is related to the front end.

By entering each developer's changes into the trained model, we can find out the programming fields he does well in.

3.4 Analysis of Contribution

We define the indicator *dcon* to evaluate the contribution of each developer to the projects he participated in. It can show the importance of him to the team.

Metrics, such as the commit frequency, lines of changed code, and number of resolved issues, are used by many employers to evaluate their employees'

Table 1. The topics and their keywords of the LDA model

Topic id	Keywords of the topic
1	string, span, class, public, cm, int, import, type, return, java, id, user, org, userid, pagesize, url, div, pageid, github, px
2	file, selection, false, true, line, dir, column, path, option, type, project, info, id, start, caret, state, end, provider, entry, editor
3	div, class, data, var, span, li, col, href, function, id, type, px, text, fa, option, pageid, style, return, content, model

performance [10]. In this paper, the developer's contribution is evaluated in terms of the proportion of added lines, deleted lines, and commits. We calculate three sub-indicators called *add_prop*, *delete_prop*, and *commit_prop*, respectively, indicating the developer's addition, deletion, and commit proportion in a project he participated in. The calculation method is shown in Equation (7).

$$item\_prop_j = \frac{item_j}{\sum_{i=1}^{m} item_i} \tag{7}$$

where *item* can be replaced by *add*, *delete*, and *commit*, j is the index of the current developer, and m is the number of developers in this project.

We use *dpcon* to show the contribution of the developer in a project, which can be expressed as the sum of the three sub-indicators.

$$dpcon = add\_prop + delete\_prop + commit\_prop \tag{8}$$

The contribution of the developer, which is named *dcon*, can be expressed as the average of his projects' *dpcon*.

$$dcon = \frac{\sum_{i=1}^{n} dpcon_i}{n} \tag{9}$$

where $dpcon_i$ is the current developer's contribution in the i-th project he participated in, and n is the number of projects he participated in.

3.5 Analysis of Personalized Commit Time

A personalized analysis of developers' commit time can help team leaders understand when their team members are more accustomed to developing and committing. We change the date of each commit to the day of the week, then count how many times a developer commits each day from Monday to Sunday and get a personalized analysis of each developer's working time.

3.6 Analysis of Participated Projects

Participated projects can reflect the developer's experience and level of development, which is also a perspective for evaluating developer's expertise. We firstly

evaluate the projects in terms of development efficiency, code quality, and collaboration degree, and get the score of each project. Then we use the sum of scores as an indicator measuring the developer's expertise.

Analysis of Development Efficiency Development efficiency is the simplest evaluation of the development team. After arranging a task, the leader definitely wants the team to complete the project as soon as possible.

According to the number and frequency of commits, we find the last commit time of the project during development, which is noted as $dlcommit\_time$.

The calculation of development efficiency is shown in Equation (10).

$$peff = \frac{lines}{dlcommit\_time - created\_time} \tag{10}$$

where $lines$ is the number of lines of the project, $dlcommit\_time$ is the date of the last commit during development, and $created\_time$ is the date the project is created.

Analysis of Code Quality The code quality of the project is measured using calculation method likes Equation (6), which is shown in Equation (11). The sub-indicators in the equation are about the values of a project.

$$pqua = \frac{\sum_k k\_rate' + complexity\_rate' + duplicated\_lines\_density'}{5} \tag{11}$$

Analysis of Collaboration We get the co-authors of each file in the project, count the number of files that every developer has modified together with other developers, and save the result as an adjacency matrix. The row number and column number of the matrix represent the developers' index, and the value of each element is the number of files modified together by these two developers (values on the diagonal are set to 0), as shown in Fig.1.

$$\begin{bmatrix} 0 & 5 & 3 \\ 5 & 0 & 1 \\ 3 & 1 & 0 \end{bmatrix}$$

Fig. 1. Project collaboration

We introduce collaboration *average* and collaboration *variance* to quantify collaboration degree of a project. The calculation of these two sub-indicators are as follows:

$$average = \frac{2\sum_i c_i}{n(n-1)} \tag{12}$$

where c_i is the i-th element at the top right of the diagonal in the matrix, and n is the number of developers of the project.

$$variance = \frac{2\sum_i (c_i - average)^2}{n(n-1)} \tag{13}$$

where c_i is the i-th element at the top right of the diagonal in the matrix, $average$ is the collaboration average of the project, and n is the number of developers of the project.

The $average$ and $variance$ are normalized using Equation (1) to obtain $average'$ and $variance'$ respectively. Finally, the collaboration indicator of the project, which is called $pcol$, is introduced.

$$pcol = average' - variance' \tag{14}$$

Project Evaluation The indicators $peff$, $pqua$ and $pcol$ are normalized to [0, 1] using Equation (1). Then the normalized indicators are calculated to obtain a comprehensive score of the project. As shown in Equation (15).

$$pscore = \frac{peff' + 1 - pqua' + pcol'}{3} \tag{15}$$

Finally, considering the number and the quality of the projects together, the evaluation indicator $dsop$ is defined, and the calculation is shown in Equation (16).

$$dsop = \sum_i pscore_i \tag{16}$$

where $pscore_i$ is the score of the i-th project that the current developer participated in.

3.7 Comprehensive Evaluation of Developers

We select code quantity, code quality, contribution and score of projects they participated in to comprehensively evaluate developers. We normalize the four indicators, which are called $dloc$, $dqua$, $dcon$, $dsop$, to [0, 1] using Equation (1). Finally, the normalized indicators are weighted and summed to obtain the comprehensive score of each developer.

$$dscore = \alpha * dloc + \beta * (1 - dqua) + \gamma * dcon + (1 - \alpha - \beta - \gamma) * dsop \tag{17}$$

$0 < \alpha$, β, γ, $1 - \alpha - \beta - \gamma < 1$ in the equation are weight parameters, and here we set $\alpha = \beta = \gamma = 0.25$.

4 Use Cases

We collected the data from our laboratory's GitLab system, and got the information of 132 developers, involving in 49 projects. Using the data, we designed and implemented an online tool to show the results of our work.

The developer's page demonstrates the analysis results of each developer in terms of development quantity, contribution, personalized commit time, skills, and code quality, which is shown in Fig.2.

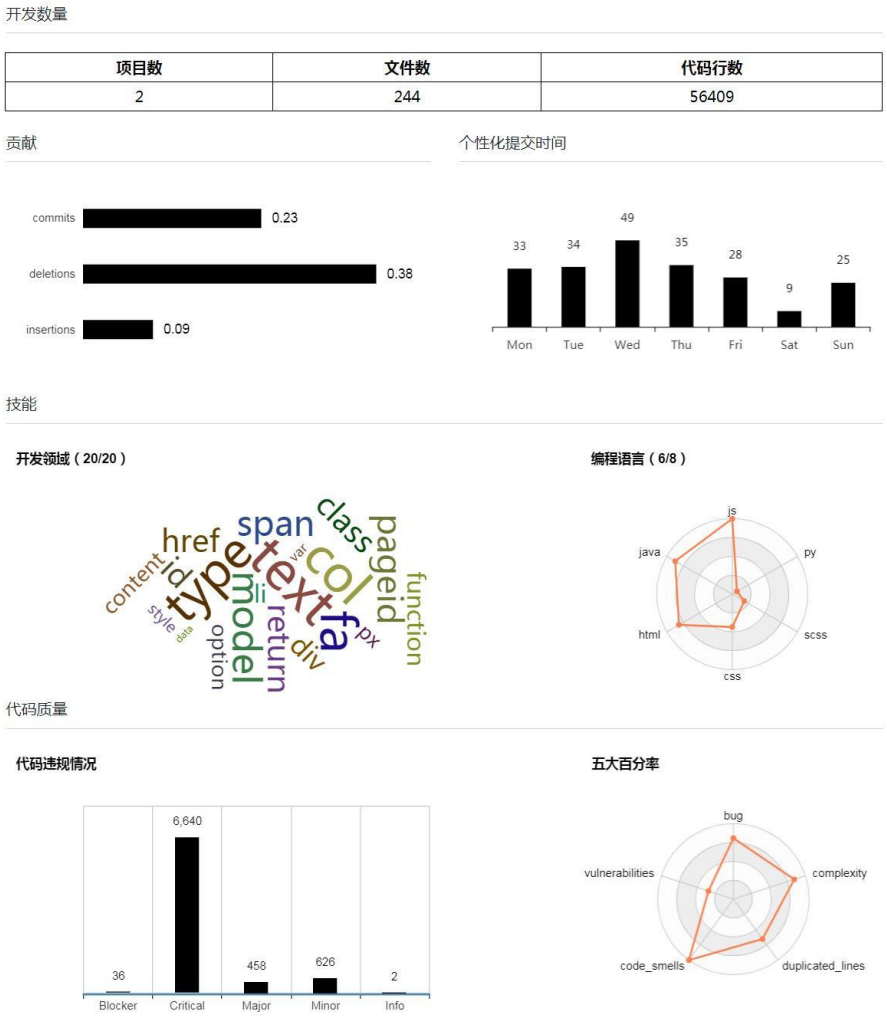

Fig. 2. The developer's page

We compared our method with a method using ncloc as the only metric to evaluate developers. Since ncloc used to be popular. The result is shown in Fig.3. For developers whose id is 7, 2, 39, 21, 14, their ncloc values are very small, indicating that their code sizes are small. Actually, most of them are new students who mainly changed some code in a few good projects. And their scores

are in the top seven using our method. In other words, our method considers four indicators when evaluating developers, which is more reasonable.

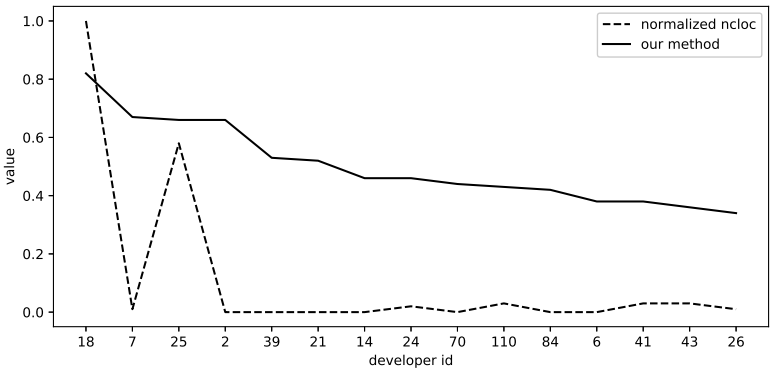

Fig. 3. Methods comparison

5 Conclusion

Evaluating the developers' expertise is a meaningful but challenging task. In this paper, we study this task based on GitLab code repositories. First, we analyze developers from six perspectives: code quantity, code quality, skills, contribution, personalized commit time, and projects they participated in. Then we use code quantity, code quality, contribution and the score of projects they participated in to evaluate developers' expertise. Finally, we develop an online developer profiling tool to show the results. Currently, our tool has been used in Neusoft and Wonders Group.

Our work can reasonably evaluate developers' expertise and rank developers according to the comprehensive scores, which provides a reference for developers' employment and promotion, as well as task assignment in the team.

There is still room for improvement in our work. The values of weights in Equation (17) are fixed and all indicators are of equal importance. The webpage will be designed to let the users offer the values of α, β and γ to show their different emphasis on different indicators. Besides, the calculation methods are relatively simple. In the future, more scientific methods will be studied to make the indicators more reasonable.

References

1. Marlow, J., Dabbish, L.: Activity traces and signals in software developer recruitment and hiring. In: Proceedings of the 2013 Conference on Computer Supported Cooperative Work, CSCW, pp. 145–156. ACM, New York (2013).
2. Thomas, F., Gail, C.M., Emily, H.: Does a programmer's activity indicate knowledge of code? In: Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ESEC-FSE, pp. 341–350. ACM, New York (2007).
3. Constantinou, E., Kapitsaki, G.M.: Developers expertise and roles on software technologies. In: 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), pp. 365–368 (2016).
4. Kagdi, H., Hammad, M., Maletic, J.I.: Who can help me with this source code change? In: 2008 IEEE International Conference on Software Maintenance, pp. 157–166 (2008).
5. Kobayakawa, N., Yoshida, K.: How github contributing.md contributes to contributors. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), vol. 1, pp. 694–696 (2017).
6. Gousios, G., Kalliamvakou, E., Spinellis, D.: Measuring developer contribution from software repository data. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR, pp. 129–132. ACM, New York (2008).
7. Li, J., Liu, J., Wu, Z., He, L.: Evaluation method of developers in GitHub based on fuzzy analytic hierarchy process. Application Research of Computers 33(1), 141–146 (2016).
8. Ke, Q, ,Wu, S: Evaluation of developer efficiency based on improved dea model. Computer & Telecommunication (6), 60–62 (2017).
9. Constantinou, E., Kapitsaki, G.M.: Identifying developers' expertise in social coding platforms. In: 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 63–67 (2016).
10. Baltes, S., Diehl, S.: Towards a theory of software development expertise. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE, pp. 187–200. ACM, New York (2018).
11. Raibulet, C., Fontana, F.A.: Collaborative and teamwork software development in an undergraduate software engineering course. Journal of Systems and Software, 144, 409–422 (2018).
12. Kosti, M.V., Ampatzoglou, A., Chatzigeorgiou, A., Pallas, G., Stamelos, I., Angelis, L.: Technical debt principal assessment through structural metrics. In: 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 329–333 (2017).
13. Blei, D.M., Ng A.Y., Jordan M.I.: Latent dirichlet allocation. Journal of machine Learning research 3(Jan), 993–1022 (2003).