CPDScorer: Modeling and Evaluating Developer Programming Ability across Software Communities

Weizhi Huang, Wenkai Mo, Beijun Shen‡, Yu Yang, Ning Li
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University, Shanghai, China
Email: zhizi1993321@163.com, jirachikai@126.com, ‡bjshen@sjtu.edu.cn, 13272436992@163.com, lnkkk1994@163.com

***Abstract*—Since developer ability is recognized as a determinant of better software project performance, it is a critical step to model and evaluate the programming ability of developers. However, most existing approaches require manual assessment, like 360 degree performance evaluation. With the emergence of social networking sites such as StackOverflow and Github, a vast amount of developer information is created on a daily basis. Such personal and social context data has huge potential to support automatic and effective developer ability evaluation. In this paper, we propose CPDScorer, a novel approach to modeling and scoring the programming ability of developer through mining heterogeneous information from both Community Question Answering (CQA) sites and Open-Source Software (OSS) communities. CPDScorer analyzes the answers posted in CQA sites and evaluates the projects submitted in OSS communities to assign expertise scores to developers, considering both the quantitative and qualitative factors. When modeling the programming ability of developer, a programming ability term extraction algorithm is also designed based on set covering. We have conducted experiments on StackOverflow and Github to measure the effectiveness of CPDScorer. The results show that our approach is feasible and practical in user programming ability modeling. In particular, the precision of our approach reaches 80%.**

***Index Terms*—Github; StackOverflow; Programming Ability Modeling; Developer Ability Evaluation.**

I. Introduction

Software developers participate in a diversity of software communities simultaneously, such as Community Question Answering (CQA) sites and Open-Source Software (OSS) communities. For example, in StackOverflow[1], developers post questions to seek for help from other peers, and they are also able to provide valuable answers for others; in Github[2], they collaboratively develop open-source softwares by committing code to software repositories. These software communities create and store a vast amount of developer information on a daily basis. Thus they provide huge potential to model and evaluate the ability of developer effectively, which plays an important role in developer recommendation, staff training as well as software project planning.

Some approaches have been proposed to target the problem of developer ability modeling in software communities [1] [2]. However, these approaches are restrict to a single community.

‡ Corresponding Author
[1]http://stackoverow.com/
[2]https://github.com/
DOI reference number: 10.18293/SEKE2016-012

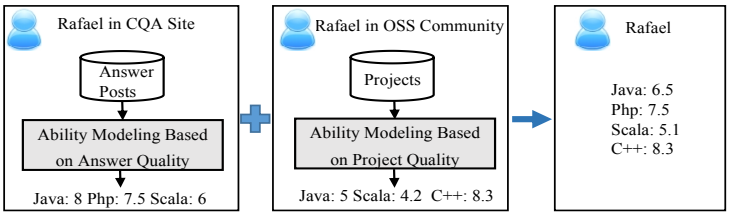


Figure 1: An Example of Developer Ability Modeling across Two Software Communities

For instance, in CQA sites, user ability can be reflected by his or her answer quality. Generally, users who provide high quality answers are more likely to own high expertise in specified subjects. The voting score of each answer is a kind of measure of answer quality as it is given by users according to their satisfaction towards the answer. Similarly, in OSS communities, developers who provide projects with high quality tend to be with a high ability in programming. In this paper, we consider developer ability modeling across software communities for more comprehensive and precise evaluation results. Figure 1 shows a simple example. Developer Rafael participates in both a CQA site and an OSS community. He is assigned ability scores under different programming skills in both two communities, and the ability scores from these two communities produce his final ability scores.

However, it is difficult to model and evaluate the ability of developer crossing communities as we need to link users between different communities first. Besides, integrating heterogeneous data of a developer from different communities is challenging. Moreover, how to derive the concrete programming ability terms to describe the programming ability of developer is also a problem. In this paper, we propose a novel approach CPDScorer to address these challenges. CPDScorer integrates one CQA site with one OSS community to corporately assess the programmer's ability from both answer quality analysis and project code evaluation. Specifically, CPDScorer models the programming ability scores of developer under various programming skills. To model the programming ability, an algorithm based on set covering is proposed to extract a set of programming ability terms. Then every answer and project are labeled with an ability term and scored based on their quality. Given answer-based ability scores in CQA site and code-based ability scores in OSS

community under different ability terms, we combine these scores by taking a weighted sum of them as the final ability scores of developers.

This paper makes the following contributions: (1) We propose an approach to modeling and evaluating the programming ability of developer crossing two software communities. Specifically, we combine answer quality evaluation in a CQA site and code quality analysis in an OSS community to model the ability of developer more comprehensively. (2) We design a programming ability term extraction algorithm based on set covering to model ability scores and ability terms jointly.

II. Related Work

A. Developer Ability Modeling in Communities

To model and evaluate the ability of developer in communities, several methods have been proposed. Zhang et al. proposed a measure called Z-score that combined user's asking and replying patterns to rate the expertise of user [3]. Liu et al. proposed CQARank to model each user's ability as expertise scores under various topics by combining textual content learning with link analysis in StackOverflow [1]. Venkataramani and Asadullah modeled the expertise of developers in a target domain by mining their activities in different open-source projects [2]. John and Gail presented an empirical evaluation of two approaches for determining the implementation expertise of developer from the data in source and bug projects [4]. All these methods mentioned above only analyze information from a single software community, and can not utilize data across different communities.

B. Answer Quality Evaluation in CQA Sites

There are some researches to evaluate the answer quality in CQA sites. Jeon et al. presented a model to predict the quality of answers based on a set of non-textual features extracted from answers, such as click counts, answerers acceptance ratio and answer length [5]. Agichtein et al. casted the problem of answer quality ranking as a binary classification problem and proposed a classification framework of estimating answer quality based on content-based features and usage-based features [6]. Shah et al. used 13 different criteria to assess the overall quality of answer and expanded the prediction model by extracting several features of the questions, the answers, and the users who provided them [7]. When modeling the developer's ability by analyzing the answer quality in CQA sites, our work borrows the idea in [5].

C. Open-Source Project Quality Analysis

With the emergence of open-source projects, measuring and evaluating their quality has attracted a lot of attention. Jarczyk et al. developed two metrics of quality for the open-source projects in Github based on both the project popularity and how fast the reported issues were solved, and analyzed the influence of collected attributes describing project and developer on quality [8]. Different from the above approach, we focus on source code evaluation for project quality. Stamelos et al. measured quality characteristics of Linux applications by a kind of software measurement tool and compared the results with the industrial standard proposed by the tool [9]. Barkmann et al. developed tools to collect code metric data from projects and described the statistical significance of individual metrics [10]. Chawla et al. implemented five metrics such as lines of code and cyclomatic complexity to analyze a set of java programs as to judge their performance with respect to the metrics [11]. We borrow the idea from [11] to analyze the quality of project source code.

III. Approach

Our proposed approach CPDScorer can model and evaluate the programming ability of developer automatically, by analyzing data from one CQA site and one OSS community.

A. Approach Overview

The overview of CPDScorer approach is illustrated in Figure 2. It is composed of five steps: (1) *Identity Linkage* links users between a CQA site and an OSS community. (2) *Programming Ability Term Extraction* extracts a set of programming ability terms based on set covering. Meanwhile, each answer or project is assigned an ability term by its topic. (3) *Answer-Based Ability Scoring* analyzes the quality of each answer in the CQA site, and assigns ability scores to the question repliers by ability terms of answers. (4) *Code-Based Ability Scoring* evaluates the quality of each software project in OSS community using code analysis technology, and scores the developers by specific project ability terms. (5) *Ability Score Composing* takes a weighted sum of answer-based ability scores and code-based ability scores as the final programming ability scores of developers.

B. Identity Linkage across Software Communities

There are already several methods proposed to link users in different software communities [12] [13]. However, these methods only considered username and email of a user while ignoring other important information like programming skills. As we need to link users between a CQA site and an OSS community, we adopt our novel tagging-based approach TBIL [14]. TBIL first extracts three kinds of features from usernames, user skills and user concerned topics from the text information of users. Based on these features, it then applies a machine learning method, Decision Tree, to obtain probabilities for every two users in different communities, which represents how likely they refer to a same person. Finally, based on these probabilities, we use a heuristic greedy matching algorithm to link these users with one-to-one constraints.

C. Programming Ability Term Extraction

After linking developers between a CQA site and an OSS community, we then need to define a set of programming ability terms to specifically describe the programming ability of developers. In some software communities, software documents are labeled with tags. These tags can be treated as ability terms because they all refer to terms about software engineering. We propose a programming ability term extraction algorithm

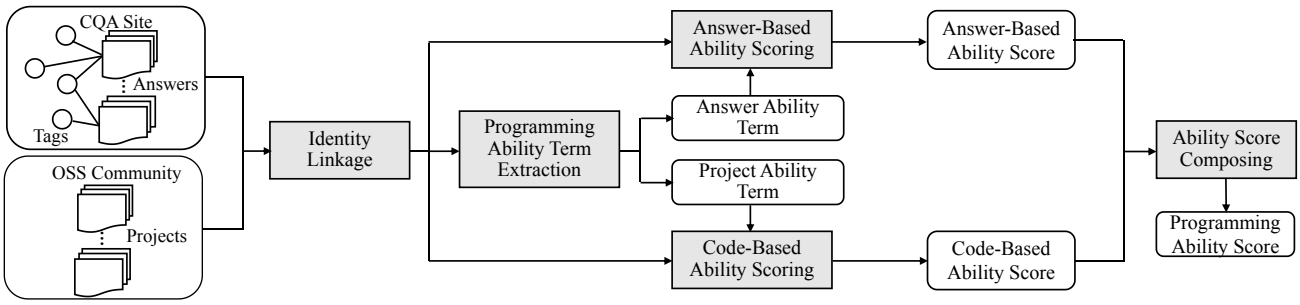


Figure 2: Overview of CPDScorer Approach

based on set covering to derive ability terms from tags. The algorithm takes a candidate programming ability term set, and a set of labeled documents (e.g., labeled answers, labeled projects, etc.) as input. It first filters out tags with frequencies less than a threshold and sorts the rest tags by their frequencies in descending order. Next, for each labeled document in the document set, if any tag of the document equals a certain candidate ability term, the ability term will be contained in the final ability term set and this document is set as covered. Finally, if the percentage of covered documents among all the documents (covering rate) reaches 90% or higher, we output the final ability term set. Otherwise, the threshold will be automatically adjusted until the covering rate is higher than 90%.

To perform the algorithm, we first need to construct a candidate ability term set. We take the set of tags in a community as the initial ability term set. Then we remove tags which are unrelated with programming ability and form a new set of tags as the candidate ability term set. For example, the tags 'excel' and 'firefox' can not reflect the programming ability of developer. As the removing process is done manually, we will clean some low frequency tags before the process to improve performance. When constructing the candidate ability term set in two different communities, there are three cases: (1) If both two communities do not have the tagging system, then we can choose the tags in StackOverflow because StackOverflow has a relatively mature tagging system with more than 38,000 diverse tags. (2) If only one community has the tagging system, tags in this community are used. (3) If both two communities have the tagging system, we select the intersection of tags in these two communities.

The detailed description of the ability term extraction procedure is shown in Algorithm 1. It takes a candidate ability term set S, and a labeled document set D as input. The algorithm first initializes the covering rate μ to 0 and the threshold θ to 5000. In each repeat, θ will be gradually reduced by 300 until μ is higher than 90%. Then our algorithm filters out some tags of which the frequencies are less than the threshold and sorts tags by their frequencies in descending order (Lines 1-6). For each labeled document, if any of its tags equals a certain candidate ability term, the status of this document is set as covered. After traversing every document, we calculate and update the covering rate (Lines 7-16). Finally, the algorithm outputs the final ability term set (Line 17).

Algorithm 1 Programming Ability Term Extraction

Input:
candidate programming ability term set S;
labeled document set $D = \{d_1, d_2, \ldots, d_N\}$;
Output:
programming ability term set H;

```
1:  initialize covering rate μ to 0
2:  initialize threshold θ to 5000
3:  repeat
4:    automatically adjust θ
5:    select the tags in S whose frequencies are larger than
      θ to form a new candidate set T = {t_1, t_2, ..., t_M}
6:    sort T by the frequencies of tags in descending order
7:    for each d_n in D do
8:      for each t_i in T do
9:        if one of the tags in d_n equals t_i then
10:         set the status of d_n as covered
11:         put t_i in H
12:       end if
13:     end for
14:   end for
15:   calculate and update μ
16: until μ is larger than 90%
17: return H;
```

Furthermore, we manually divide the programming ability terms into five categories, including ability terms about programming language (such as java, php, etc.), database (such as mysql, mongodb, etc.), framework (such as asp.net, spring, etc.), library (such as jquery, backbone.js, etc.) and others (such as Android, etc.).

D. Answer-based Ability Scoring

High quality answers are expected to reflect that the replier has a good knowledge of the specific programming domain (described by the programming ability term) the answers belong to. So, the first sub-step is to assign the appropriate ability term to every answer by matching its tags with terms in the programming ability term set. However, answers in some CQA sites are not labeled with any tags. For such answers, we extract keywords from the content of each answer using

TextRank algorithm [15]. And we match these keywords with programming ability terms to select a most representative term for the answer.

Then the second sub-step is to analyze the quality of each answer in CQA site, and work out the ability scores of the replier. In most CQA cites, answers can be voted up or down considering their quality. Users can also comment on them. We extract features determining answer quality from each answer post, such as answer length, the number of upvotes, and the number of downvotes. After scoring each answer based on its features, the ability score for developer u under a specific programming ability term t is the average score of answers posted by the developer of which the ability term is just t, which is defined as:

$$SEScore(u,t) = \frac{\sum_{ans \in Answer(u)} H(ans,t) AScore(ans)}{\sum_{ans \in Answer(u)} H(ans,t)} \quad (1)$$

where $AScore(ans)$ denotes the evaluation score of answer, $Answer(u)$ represents all the answers posted by the developer u and $H(ans,t)$ is an indicator function, returning 1 if the ability term of answer is t.

E. Code-based Ability Scoring

To model the ability scores of developer based on project quality, we need to obtain the ability term for every project first. As some OSS communities do not have a tagging system, we extract keywords from the description files of projects like README files in Github using TextRank algorithm [15]. The extracted keywords will later be used to match the ability terms for the project ability term. Then we use static code analysis tool Understand to extract code metrics (e.g., CountLineCode, AvgLine, CountPath, etc.) and score projects with code metrics.

Given the score of one project p a developer u involved in, we define Equation (2) to calculate the ability score $GEScore(u,t)$ of the developer under a specific project term t, where $I(p,t)$ is an indicator function, returning 1 if the ability term of project is t. We use $P(u)$ to denote all the projects that developer u are involved in and we define all the developers in the project as $U(p)$. $PScore(p)$ denotes the evaluation score of project p. The contribution that developer u makes to project p is defined as $Cont(u,p)$, which is provided in our dataset. If the contribution of developers is not offered, we can measure it by their commit frequency in the project.

$$GEScore(u,t) = \sum_{p \in P(u)} I(p,t) PScore(p) \frac{Cont(u,p)}{\sum_{u \in U(p)} Cont(u,p)} \quad (2)$$

The project score is allocated to developers according to their contribution to the project. The more contribution they make to the project, the higher score they are allocated. For the developer's ability score under a certain skill t, we sum up all the scores of projects with the target ability term t that the developer is involved in.

https://scitools.com/

F. Ability Score Composing

Given a target developer u, *Ability Score Composing* combines the two ability scores along with their programming ability term t produced by *Answer-Based Ability Scoring* and *Code-Based Ability Scoring* into a unified expertise score. We define the final expertise score of developer u in programming skill t denoted by $EScore(u,t)$ as follows:

$$EScore(u,t) = \alpha \times SEScore(u,t) + \beta \times GEScore(u,t) \quad (3)$$

where $SEScore(u,t)$ and $GEScore(u,t)$ are the ability scores of programming skill t evaluated by the two scoring components respectively, and $\alpha, \beta \in [0,1]$ are weights assigned to the two scores.

IV. Experiments and Results

In this section, we conduct experiments to validate the performance of CPDScorer. We first present our experimental settings and then analyze the results of experiments.

A. Experimental Settings

As StackOverflow is one of the most famous CQA sites and Github is a representative OSS community, we select them to conduct the experiments. The data of StackOverflow and Github is before January, 2015. There are totally 3,106,381 posts including 255,401 answer posts, and 32,207 tags. Besides, there are totally 5791 linked developers involved in 133,895 projects. We extract 305 programming ability terms from the tags in StackOverflow. And the extracted answer features and code metrics are shown in Table I and Table II. There are 5,230 answers and 2,500 projects in training data rated by five master students whose major is computer science on scale 1 to 10. As CPDScorer can model the developer's ability scores under each ability term, we select the top-10 developers in each domain by their ability scores. Then, we ask another five students to evaluate and rate our results by giving 'yes' or 'no' to show whether they are satisfied with the results. Specifically, they will investigate the detailed profiles of the corresponding developers in both StackOverflow and Github, including the quality of their answers, the quality of their projects, their activity of contribution and other related information, to determine whether the results are reasonable. The precision of the CPDScorer approach is defined as the following equation, where M represents the number of 'yes' and N denotes the total number of results (the number is 10 in our experiments).

$$Precision = \frac{M}{N} \quad (4)$$

B. Developer Ability Modeling Using Different Regression Methods

We adopt two widely used machine learning methods: Linear Regression and Regression Tree M5P [16], and separately apply these two methods to both answer features and code metrics. Thus there are four combinations of methods, as shown in Table III. To compare the final results of ability scores using different combinations of methods for their effectiveness, we

TABLE I: LIST OF FEATURES

Features	Description	Coefficients
Answer Length	The length of the answer	0.0023
Answer Vote Score	The score of the answer is calculated for based on the number of upvotes and downvotes of the answer	0.0781
Comment Count	The number of comments to the answer	0.0113
Answer Acceptance	Whether the answer is accepted by the question owner is a direct feedback on the quality of the answer	0.2388
Number of DownVote	The number of the DownVote the answer receives, which shows the answer is useless	-0.019
Number of UpVote	The number of the UpVote the answer receives, which shows the answer is useful	0.0041
Number of Answers	The number of answers for the given programming topic	0.0038

TABLE II: LIST OF CODE METRICS

Code Metrics	Description	Coefficients
AvgCyclomatic	Average cyclomatic complexity for all nested functions or methods	-3.9903
AvgCyclomaticModified	Average modified cyclomatic complexity for all nested functions or methods	4.8879
AvgCyclomaticStrict	Average strict cyclomatic complexity for all nested functions or methods	-0.6188
AvgLineBlank	Average number of blank for all nested functions or methods	-0.4428
AvgLineCode	Average number of lines containing source code for all nested functions or methods	0.8015
CountDeclFunction	Number of functions	0.7291
CountLineBlank	Number of blank lines	-0.845
CountLineCode	Number of lines containing source code	-1.6516
CountLineCodeExe	Number of lines containing executable source code	-0.4279
CountStmtExe	Number of executable statements	9.4282
Cyclomatic	Cyclomatic complexity	0.4577
SumCyclomaticStrictv	Sum of strict cyclomatic complexity of all nested functions or methods	-7.4112

randomly select three ability terms from each ability category and pick the top-10 developers with each ability term to be rated by those five students every time. And the process will be repeated five times. Furthermore, we take the average precision of score results under all ability terms from a specific category as precision of the category. However, we exclude the 'other' category on account that the number of developers ranking top can't reach 10 for most ability terms from that category.

TABLE III: COMBINATION OF DIFFERENT METHODS

	Linear Regression for AF	**M5P for AF**
M5P for CM	M5P×LR	M5P×M5P
Linear Regression for CM	LR×LR	LR×M5P

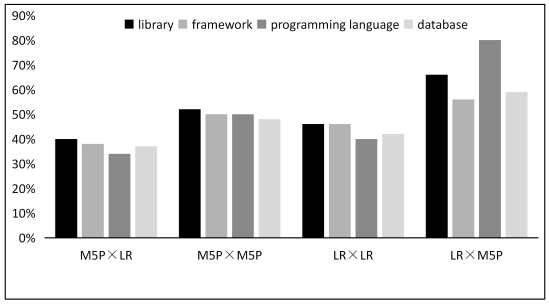
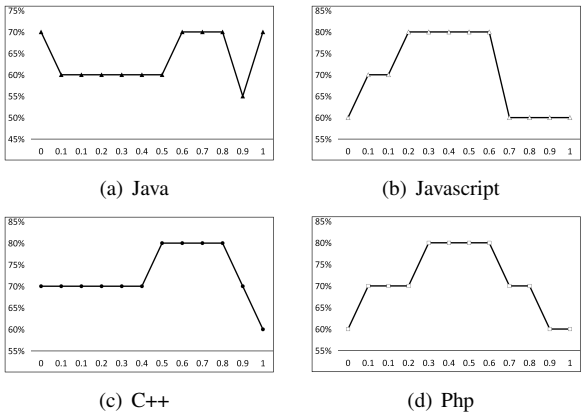


Figure 3: Results of Different Combinations of Methods

Figure 3 depicts the results. In Figure 3, the horizontal axis represents the combinations of different methods and the vertical axis denotes their respective precisions in the four categories. From the results, we can see the combination LR×M5P achieves the best performance with the precision of 80% among the four combinations in all categories followed by M5P×M5P while LR×M5P has the worst precision. Thus in our experiment, we apply the M5P to answer features and Linear Regression to code metrics.



(a) Java (b) Javascript

(c) C++ (d) Php

Figure 4: Precisions at Different Values of α

C. Features Contributions

When applying M5P to answer features and Linear Regression to code metrics, it has the best performance. Thus only the coefficients of answer features for M5P and coefficients of code metrics for Linear Regression are presented and discussed.

From Table I, it is not surprising the features 'Answer Vote Score' and 'Answer Acceptance' have a significant positive effect on the outcome quality scores, while 'Number of Downvote' has a negative effect. When the questioner accepts one answer, it means the answer works for him or her well. The larger the 'Answer Vote Score' is, the higher value the answer will be thought of. Conversely, 'Number of Downvote' suggests that users consider the answer is useless.

As shown in Table II, more than half features have negative impacts on the quality of project source code. Most of them do not affect the performance by much. However, the features 'AvgCyclomatic', 'CountStmtExe', and 'SumCyclomaticStrict' have significant contribution to the code quality.

D. Parameter Sensitivity Analysis

In this experiment, we vary the weights α and β assigned to ability scores from the two communities to understand the impact of varying their values on the precision. The sum of α and β is always 1.0, so we just adjust the value of α. We start by initializing α to 0. Then, we incrementally increase the value of α by 0.1 until it reaches 1. For each combination of α and β, we evaluate the top-10 developers. As the number of ability terms about programming language accounts for nearly half of all ability terms, we focus on analysing scores under programming languages.We randomly select four programming language, such as java, javascript, c++ and php.

Figure 4 illustrates the precisions at different values of α. In Figure 4, horizontal axis denotes different values of α, ranging from 0 to 1. The values in vertical axis reflect the precision of corrsponding α. The presicion of CPDScorer remains quite stable across a wide range of parameters α. The best precision is achieved among the four ability terms when the value of α becomes 0.6. As a result, we choose the value 0.6 for α and the value 0.4 for β in our work.

V. Conclusion and Future Work

In this paper, we propose a novel approach called CPDScorer for modeling and evaluating the programming ability of developers crossing two communities. CPDScorer considers both answer quality about programming topics in a CQA site and project source code quality in an OSS community. A programming ability term extraction algorithm is also designed to label every answer and project with an ability term. Two famous communities, StackOverflow and Github, are selected to validate the feasibility of our approach.

To evaluate the programming ability of developers more accurately, there are still some aspects for improvement in the future: (1) When modeling the developer's ability by estimating their answer posts, we will extract more features such as the answer comment content. In particular, the profile of the replier may play an important role in answer quality. (2) When modeling the programming ability in OSS communities, we can explore other factors such as the commit messages of developer for project quality evaluation.

Acknowledgement

This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242).

References

[1] L. Yang, M. Qiu, S. Gottipati, F. Zhu, J. Jiang, H. Sun, and Z. Chen, "Cqarank: jointly model topics and expertise in community question answering," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pp. 99–108, ACM, 2013.

[2] R. Venkataramani, A. Gupta, A. Asadullah, B. Muddu, and V. Bhat, "Discovery of technical expertise from open source code repositories," in *Proceedings of the 22nd international conference on World Wide Web companion*, pp. 97–98, International World Wide Web Conferences Steering Committee, 2013.

[3] J. Zhang, M. S. Ackerman, and L. Adamic, "Expertise networks in online communities: structure and algorithms," in *Proceedings of the 16th international conference on World Wide Web*, pp. 221–230, ACM, 2007.

[4] J. Anvik and G. C. Murphy, "Determining implementation expertise from bug reports," in *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*, pp. 2–2, IEEE, 2007.

[5] J. Jeon, W. B. Croft, J. H. Lee, and S. Park, "A framework to predict the quality of answers with non-textual features," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 228–235, ACM, 2006.

[6] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne, "Finding high-quality content in social media," in *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pp. 183–194, ACM, 2008.

[7] C. Shah and J. Pomerantz, "Evaluating and predicting answer quality in community qa," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pp. 411–418, ACM, 2010.

[8] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki, "Github projects. quality analysis of open-source software," in *Social Informatics*, pp. 80–94, Springer, 2014.

[9] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, "Code quality analysis in open source software development," *Information Systems Journal*, vol. 12, no. 1, pp. 43–60, 2002.

[10] H. Barkmann, R. Lincke, and W. Lowe, "Quantitative evaluation of software quality metrics in open-source projects," in *Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on*, pp. 1067–1072, IEEE, 2009.

[11] M. K. Chawla and I. Chhabra, "Implementing source code metrics for software quality analysis," in *International Journal of Engineering Research and Technology*, vol. 1, ESRSA Publications, 2012.

[12] S. Liu, S. Wang, F. Zhu, J. Zhang, and R. Krishnan, "Hydra: Large-scale social identity linkage via heterogeneous behavior modeling," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 51–62, ACM, 2014.

[13] E. Kouters, B. Vasilescu, A. Serebrenik, and M. G. van den Brand, "Who's who in gnome: Using lsa to merge software repository identities," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pp. 592–595, IEEE, 2012.

[14] W. Mo, B. Shen, Y. Chen, and J. Zhu, "Tbil: A tagging-based approach to identity linkage across software communities," in *Asia-Pacific Software Engineering Conference*, pp. 56–63, IEEE, 2015.

[15] A. A. Abbasi and M. Younis, "A survey on clustering algorithms for wireless sensor networks," *Computer communications*, vol. 30, no. 14, pp. 2826–2841, 2007.

[16] W. Han, L. Jiang, T. Lu, and X. Zhang, "Comparison of machine learning algorithms for software project time prediction," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 10, no. 9, pp. 1–8, 2015.