Recommending Crowdsourced Software Developers in Consideration of Skill Improvement

Zizhe Wang, Hailong Sun*, Yang Fu, Luting Ye
State Key Laboratory of Software Development Environment
School of Computer Science and Engineering, Beihang University
Beijing, China
{wangzz,fuyang,yeluting}@act.buaa.edu.cn
sunhl@buaa.edu.cn

Abstract—Finding suitable developers for a given task is critical and challenging for successful crowdsourcing software development. In practice, the development skills will be improved as developers accomplish more development tasks. Prior studies on crowdsourcing developer recommendation do not consider the changing of skills, which can underestimate developers’ skills to fulfill a task. In this work, we first conducted an empirical study of the performance of 74 developers on Topcoder. With a difficulty-weighted algorithm, we re-compute the scores of each developer by eliminating the effect of task difficulty from the performance. We find out that the skill improvement of Topcoder developers can be fitted well with the negative exponential learning curve model. Second, we design a skill prediction method based on the learning curve. Then we propose a skill improvement aware framework for recommending developers for software development with crowdsourcing.

Index Terms—Crowdsourcing, recommender systems, software development, Topcoder.

I. INTRODUCTION

Crowdsourcing has become a prevalent paradigm for software development. Platforms such as Topcoder, Freelancer and Kaggle usually adopt a contest mechanism to elicit contributions from individual developers. Take Topcoder as an example, a requestor publishes a task to Topcoder with task description and a certain amount of monetary rewards. Then developers on Topcoder platform check the task description and the rewards to see if it is worthwhile doing the task. Next, the interested developers will register with the task and submit their deliverables to Topcoder after they finish the development. Then a core team on Topcoder is responsible for evaluating the submissions from developers. Some top ranked developers are granted rewards according to the pre-set rules.

The above process is also known as the open-call mode, in which the software development productivity and quality depend heavily on the skills and efforts of the developers who answer the task call. Recently, several efforts [1], [2] are made to recommend developers to a given task. Essentially, existing work concerns finding developers whose skills can meet the requirements of a task. In practice, software development skills are gradually improved as developers perform more tasks. However, no efforts are seen to study the skill improvement issue in crowdsourced software development. Instead, existing work on developer recommendation treats the skills of developers statically.

In this work, we aim at studying a developer recommendation approach for crowdsourced software development by considering the skill improvement of developers. To achieve this goal, we choose Topcoder as our research platform because Topcoder is known as one of the earliest and largest crowdsourcing platforms. Note: In Topcoder a task is also referred to as a challenge. We interchangeably use the two terms in the rest of this paper. To be specific, we are mainly concerned with answering three research questions:

• How to measure the skill level of developers? Intuitively, in Topcoder this can be realized by using the scores that a developer obtains for finishing a development task. However, received scores are also affected by the difficulty of tasks. Thus, the raw score data cannot be straightforwardly used to measure a developer’s skills.

• How do the skills of developers improve over time? Intuitively, a developer will improve her skills as she accomplishes more and more challenges. However, a clear understanding to the improvement is still a pending issue.

• What can we do when we know the way that skills improve? This is the ultimate question we want to answer in this study. Actually, we aim at designing an approach to recommend developers by considering the skill improvement.

To answer these questions, we selected 74 developers who have participated over 100 challenges in Topcoder. And for each developer, the score of each submission is available. First, to measure the skills of a developer, we design a difficulty-weighted scoring algorithm. In this algorithm, we first define the difficulty of challenges as a synthesized score of four attributes including the time of duration, prize, number of registrants and reliability bonus. The four attributes of each submission are all given in Topcoder. Then the scores of each submission are weighted according to the difficulty. Second, we model the skill improvement with known learning curves [3], [4] with the statistical analysis of the difficulty-weighted scores of each developer. And we find that the negative exponential learning curve can best model the skill improvement of Topcoder developers. Third, we design a skill improvement aware
framework for recommending developers for crowdsourced software development tasks, where the learning curves of developers are incorporated into recommendation algorithms. The major contributions of this work are as follows:

- We have empirically studied the skill improvement issue in Topcoder and we confirm that developers do learn as they perform more development tasks. As far as we know, we are the first to study this problem in crowdsourced software development.
- We propose to model the skill improvement with learning curve models and verify that the negative exponential learning curve is suitable for characterizing the skill improvement of developers in Topcoder.
- We present an approach to recommend developers for crowdsourced software development by incorporating skill improvement.

The remainder of this paper is organized as follows. In Section II, we propose a difficulty weighted algorithm to measure the skills of a developer. Section III presents our study of using learning curves to describe the skill improvement. Section IV gives a prediction model with the learning curve. In Section V, we design a recommendation approach considering skill improvement. Section VI discusses the related work. Finally in Section VII, we conclude this work.

II. REPRESENTING DEVELOPERS’ PERFORMANCE WITH DIFFICULTY-WEIGHTED SCORES

A. Data Preparation

We crawled a dataset from Topcoder involving 32,565 challenges, 7,620 developers who made 59,230 submissions from 2006 to 2016. For most developers, the number of submissions are less than 10. And we chose 100 developers who have over 100 submissions to study the changing of the developers’ skills.

A final score is given when a submission is submitted on Topcoder. When a submission’s final score is 0, it usually means the submission is unfinished, which cannot reflect the skill level of the developer. So we delete the submissions which score 0. Then there are finally 74 developers left.

Our dataset and source code can be found on GitHub\(^1\).

B. Correlation Analysis

To characterize developers’ performance, we first directly plot the final scores of each developer’s submissions in chronological order. As shown in Figure 1, there are no apparent trends in developers’ performance.

We further conducted a Spearman correlation analysis between the final scores and the chronological order of those scores, and the results are presented in Figure 2.

In Figure 2, we find that the majority of Spearman correlation coefficients are between −0.3 and 0.3. The mean value of \(|r|\) is 0.3155 and the mean value of \(r\) is 0.2639. Therefore we can conclude that the correlation between performance and time is weak.

\(^1\)https://github.com/crowdintelligence/LCTopcoder

However, this result does not comply with our intuition of the skill improvement with time. Thus we are wondering whether the final scores can reflect the true skills of developers? A straightforward thought is that the final scores are also affected by the difficulty of challenges except the development skills. Developers usually get higher scores on less difficult challenges. And the results shown in Figure 2 can be caused by the un-uniform distribution of the challenge difficulty. Next we analyze the factors that may reflect the challenge difficulty.

C. The Factors Reflecting Difficulty

Given a challenge, we consider four parameters that can be used to infer the difficulty of challenges, including the time of duration, prize, number of registrants and reliability bonus.

- **Time of duration**: Usually, a difficult and complex challenge needs more time to be processed.
- **Prize**: The prize given by the requester is also a significant attribute. A high prize means the challenge is more difficult and needs developers with better skills.
- **Number of registrants**: The number of registrants is the number of developers who register with the challenge. A large number of registrants mean it is more difficult to win the prize.

Fig. 1. Scatter diagram of 8 developers’ final score. X axis represents the serial number of time, Y axis is the final scores.

Fig. 2. The distribution of 74 developers’ Spearman correlation coefficients.
Reliability bonus: Reliability bonus is the bonus that the requester sets to ask for a more reliable program. The difficulty increases when the reliability is asked for.

### D. The Algorithm

We first normalize the four attributes to [0, 1]. For challenge $i$, we denote its time of duration by $t_i$, and denote the maximum of $t_i$ by $t_{\text{max}}$. Then the time parameter of challenge $i$ is represented as follows:

$$T_i = \frac{t_i}{t_{\text{max}}}. \quad (1)$$

In a similar way, we define the prize parameter $P_i$, registrant parameter $R_i$, and reliability parameter $B_i$ as $P_i = \frac{p_i}{p_{\text{max}}}$, $R_i = \frac{r_i}{r_{\text{max}}}$, and $B_i = \frac{b_i}{b_{\text{max}}}$ respectively.

Since the four attributes are all positively related to difficulty, the summation of them is also positively related to difficulty. So we define the difficulty parameter $D_i$ as the summation of the four parameters.

$$D_i = T_i + P_i + R_i + B_i. \quad (2)$$

Multiplying the final scores by the difficulty parameters, we get the difficulty-weighted scores, which eliminate the influence of challenge difficulty. By arranging the weighted scores of each developer according to time, we can observe an apparent growing trend of developers’ skills. Figure 3 plots the results of 8 developers randomly selected from the 74.

Similarly, we calculate the Spearman correlation coefficient $r$ between the difficulty-weighted final scores and time using the same method. The distribution of $r$ can be seen in Figure 4. 68.92% of the developers have an $r$ greater than 0.3. And the mean value of $|r|$ is 0.4052, the mean value of $r$ is 0.3787, which shows a moderate correlation. So we can conclude that the difficulty algorithm is effective, and the performance and time have a positive correlation.

### III. THE LEARNING CURVE MODELS OF DEVELOPERS’ PERFORMANCE

Since the positive correlation has been observed, we introduce two learning curves to fit the scatter diagram of the difficulty-weighted scores and make a comparison.

#### A. The Hyperbolic Learning Curve

Let $w$ be a developer with the prior knowledge $p_w$ and the learning speed $r_w$. And the highest score is $K$. Then, according to the hyperbolic learning curve [3], [4], the performance $Q_w(x)$ of developer $w$ can be defined as

$$Q_w(x) = K \frac{x + p_w}{x + p_w + r_w}. \quad (3)$$

Regarding the serial numbers of each developer’s submissions as $x$ and the difficulty-weighted score of these submissions as $Q_w(x)$, we can estimate the value of $p_w$ and $r_w$. To learn a model for $Q_w(x)$, we let $Z_{hw}(x) = \frac{K}{x - Q_w(x)}$, $\alpha_{hw} = \frac{1}{r_w}$, $\beta_{hw} = \frac{p_w}{r_w} + 1$.

Then we get a linear model

$$Z_{hw}(x) = \alpha_{hw}x + \beta_{hw}. \quad (4)$$

Using linear-regression method, we estimate the value of $\alpha_{hw}$ and $\beta_{hw}$. And then, we calculate the value of $p_w$ and $r_w$. Part of fitting results are shown in Figure 5 and the values of $p_w$, $r_w$ and $\text{normr}$ of them are shown in Table I. $\text{normr}$ stands for the norm of residuals. From Figure 5 and Table I, we find that the hyperbolic learning curve doesn’t fit the data well. The value of $p_w$ stands for the prior knowledge of developer $w$, and it should be positive in this model. But in Table I, some of them are negative, which indicates that the fitting is not appropriate. Since that, we adopt another learning curve model called negative exponential learning curve model to fit the experimental data.

#### B. The Negative Exponential Learning Curve

Let $w$ be a developer with the prior knowledge $p_w$ and the learning speed $r_w$. Then, according to the negative exponential
learning curve [4], the performance $Q_w(x)$ of developer $w$ can be defined as

$$Q_w(x) = K(1 - e^{-\frac{x}{r_w}}).$$  \hspace{1cm} (5)

Similarly, we transform Eq. (5) into a linear function by equations $Z_{ew}(x) = \ln(Q_w(x) - K)$, $\alpha_{ew} = \frac{1}{r_w}$, $\beta_{ew} = \frac{p_w}{r_w} - \ln K$.

So the linear model is

$$Z_{ew} = \alpha_{ew}x + \beta_{ew}.$$  \hspace{1cm} (6)

Using least square linear regression, we estimate the value of $\alpha_{ew}$ and $\beta_{ew}$. And then, we calculate the value of $p_w$ and $r_w$. Part of fitting results are shown in Figure 5 and the values of $p_w$ and $r_w$ of them are shown in Table II.

C. Comparison

From Figure 5 and Figure 6, we can see that the negative exponential learning curve fits the data better than the hyperbolic one. In Table I and Table II, the norms of the residuals of the negative exponential model are far less than the hyperbolic one. And the p-values in Table II are all less than 0.05, which means the fittings are reliable at a 95% significance level. On the contrary, some of the p-values in Table I are larger than 0.05, indicating that the fittings are not reliable.

Actually, among all the 74 fittings using the negative exponential model, there are respectively 50, 57, 61 fittings have p-values less than 0.01, 0.05, 0.1. But for the hyperbolic model, there are only 26, 40, 44 fittings have p-values less than 0.01, 0.05, 0.1. This shows that negative exponential model is more reliable to fit the difficulty weighted final scores. The results also show that most developers’ skills improvement obeys the negative exponential learning curve.

IV. THE SKILL PREDICTION MODEL

A. The Prediction Model

We propose to use the learning curve model to predict the performance of developers and recommend the developers who may outperform others for requester.

Since the negative exponential learning curve performs well in fitting the developer’s scores, we use it to predict the developers’ performance. We choose the 50 developers whose p-values are less than 0.01 in Section III, which means the learning curve law is significant on them. We select the first 100 weighted final scores of each developer, and use the first $N$ scores to estimate the value of parameter $p_w$ and $r_w$ in Eq. (5). Then we use the model as a prediction model and use the next $M$ scores to evaluate the model.
Fig. 7. The R values of the prediction model, Last One method and Mean method. X axis is N value, Y axis is R value. M=5.

B. Comparison

We design three prediction methods as comparisons to our prediction model.

- Hyperbolic: This method uses the hyperbolic learning curve to fit the 50 developers’ first N scores and employs the models to predict the next M scores.
- Mean: This method calculates the mean values of each developers’ first N scores and regards the mean values as predicted values of the next M scores.
- Last one: This method regards the Nth scores of the 50 developers as predicted values of the next M scores.

C. Evaluation

To evaluate the reliability of prediction, we calculate the difference between the predicted values and the real values of the next N scores. The equation to calculate the difference is as follows:

\[ R_{N,M} = \frac{1}{50} \sum_{w=1}^{50} \sqrt{\frac{1}{M} \sum_{i=1}^{M} (Q_w(N+i) - y_w(N+i))^2}, \]  
(7)

where \( y_w(N+i) \) is the real value of the \( (N+i)_{th} \) score.

The comparison under different N and M can be seen in Figure 7, since the \( R_{N,M} \) values are far larger than the other methods, Figure 7 shows the \( R_{N,5} \) values of Exponential, Mean, and Last one methods under different N values. We can see that our prediction model has the minimal R values when N is more than 60. Actually, our prediction model always has the minimal R values with an N larger than 60 when M changes from 1 to 10.

A lower R value means a more reliable prediction, so we can conclude that our prediction model has a superior forecasting performance.

V. THE RECOMMENDATION WORKFLOW

In prior recommendation methods, the improvement of developers’ skills are not taken into account. Thus we design a recommendation workflow on the basis of the difficulty algorithm and the learning curve model. The workflow is divided into three parts and it is illustrated in Figure 8.

A. Building Learning Curve Models

In Section IV, we have prediction models of many developers, and they perform well. But we can’t use these models for recommendation directly. Since different tasks require different skills, developers won’t do the tasks they are not good at. Thus we should select the developers who are proficient at certain kind of tasks first and build their learning curve models on different kind of tasks.

At first, we apply an algorithm such as K-Means to cluster the tasks in the historical data. For each cluster, we build learning curve models of each developer with their difficulty-weighted scores in the cluster. Then we get the learning curve models that forecast the developers’ performance on different kind of tasks.

B. Finding Developers

When a task is released, we suppose to find out which developers are good at this kind of task. We can classify the new task by calculating its distance to each cluster mentioned in Section V-A. The distance can be calculated by measuring their differences in platforms and techniques or other characteristics. The cluster with the shortest distance will be the most similar cluster to the new task. And the developers who have done the tasks in the cluster are those who are good at this kind of tasks.

C. Predicting and Ranking the Developers

Knowing who are good at this kind of task, we can find their learning curve models in this cluster. Then we can predict their possible performance on the new task by calculating the difficulty-weighted scores next time. Since the difficulty in the difficulty-weighted scores is the difficulty of the new task, the difficulty-weighted scores of each developer have the same difficulty-weight. Therefore, we can compare the performance of each developer by comparing their difficulty-weighted scores.

By ranking the scores, we get the developers who score higher and recommend these developers to the new task.
VI. RELATED WORK

Our work concentrates on the change of worker performance in crowdsourcing process. In crowdsourcing, there are a large number of works concentrating on worker ability. Some design a tool to analyze the performance data of developers for improvement recommendation [1], some identify developers of high quality from their open source contributions [2], some weigh workers according to their inferred expertise for hiring tasks may require different skills, we will further study how good worker [5], [6], [7]. Nevertheless, they don’t consider about that the performance will change over time.

There are also some works taking the change of worker ability into consideration. Some researchers use a time-serial model to describe and predict the change of worker performance [8]. And some use a hyperbolic learning curve to model the performance change[9], which is close to our work.

Studies in industry engineering indicate that the quality of workers improve when they complete repetitive work [10], [11]. And learning curve is a mature theory for describing such improvements [12]. There are many types of learning curve [3], [4]. Such as hyperbolic learning curve, exponential learning curve DeJong learning curve and S-curve. In this paper, we use the hyperbolic model and negative exponential model to fit the performance data, since they are designed to measure and predict each worker’s percentage of correct completed tasks [4], [13], which are replaced by a difficulty weighted score in our work. Hyperbolic model is proved more efficient, stable and robust than many other models [3], [4]. But in our work, the negative exponential model outperforms it.

VII. CONCLUSION

In this paper, we study how crowdsourced software developers’ skills get improved over time. Specifically, we conducted an empirical study on a dataset collected from Topcoder, one of the most popular web sites for software development with crowdsourcing. We first plot the final scores of the selected developers in chronological order and cannot obtain any meaningful observation. Then we design an algorithm to cancel off the effect of task difficulty on developers’ scores, and re-plot the difficulty-weighted scores. We find that the performance of developers does improve over time, which is consistent with the intuition. Furthermore, we aim at fitting developers’ performance with existing learning curves, and we show that the negative exponential learning curve model is better than the hyperbolic learning curve. With the negative exponential learning curve model, we can predict the future performance of developers. And we finally present a framework for recommending developers based on the prediction results with the negative exponential learning curve.

Future work leads to three directions. First, as different tasks may require different skills, we will further study how developers’ skills improve on certain type of tasks. Second, we aim at designing practical recommendation algorithms on the basis of the obtained learning curve models and conduct experimental evaluations on real-world datasets. Third, we plan to extend our study to other crowdsourced software development sites (e.g. Kaggle) and opensourced software development communities (e.g. GitHub).

ACKNOWLEDGMENT

This work was supported partly by the National Key Research and Development Program of China under Grant No.2016YFB1000804, the National Basic Research 973 Program of China under Grant Nos. 2014CB340304, 2015CB358700, and the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2017ZX-14.

REFERENCES