Developer Recommendation for Crowdsourced Software Development Tasks

Ke Mao*, Ye Yang†, Qing Wang‡, Yue Jia*, Mark Harman*
*CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
†School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ, 07030, USA.
‡Lab for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing, 100190, China.
k.mao@cs.ucl.ac.uk, ye.yang@stevens.edu, wq@itechs.iscas.ac.cn, yue.jia@ucl.ac.uk, m.harman@ucl.ac.uk

***Abstract*—Crowdsourced software development utilises an open call format to attract geographically distributed developers to accomplish various types of software development tasks. Although the open call format enables wide task accessibility, potential developers must choose from a dauntingly large set of task options (usually more than one hundred available tasks on TopCoder each day). Inappropriate developer-task matching may harm the quality of the software deliverables. In this paper, we employ content-based recommendation techniques to automatically match tasks and developers. The approach learns particular interests from registration history and mines winner history to favour appropriate developers. We measure the performance of our approach by defining accuracy and diversity metrics. We evaluate our recommendation approach by introducing 4 machine learners on 3,094 historical tasks from TopCoder. The evaluation results show that promising accuracy and diversity are achievable (accuracy from 50% to 71% and diversity from 40% to 52% when recommending reliable developers). We also provide advice extracted from our results to guide the crowdsourcing platform in building a recommender system in practice.**

I. Introduction

In recent years, an increasing number of successful software companies have turned to employ decentralised software ecosystems such as open source communities and crowdsourcing to augment their software production [1]. Crowdsourced Software Development (CSD), which derives its concept from Crowdsourcing, utilises an open call format to attract online developers to accomplish various types of software development tasks such as architecture, component design, component development, testing and bug fixing.

Jeff Howe [2] first coined the definition of crowdsourcing in 2006. Although crowdsourcing is not specially proposed for software engineering domains, its application in software development is growing. A crowdsourcing industry report from Massolution [3] indicates the number of workers engaged in software development increased by 151% in the year 2011. This increase is even more dramatic than the increase in crowdsourced micro-tasks: Amazon Mechanical Turk (AMT) is one of the most popular marketplaces for crowdsourcing micro-tasks such as photo tagging, logo design and quick idea gathering. Crowdsourcing platforms that support software development include TopCoder, uTest, GetACoder, eLance, Guru, Freelancer, Tackcn, etc. Among them, TopCoder[1] is the world's biggest competitive software development portal [4]. Its clients include Google, Microsoft, Facebook and AOL. Compared with traditional software development, TopCoder's crowdsourced development is claimed to exhibit the ability to deliver custom requested software assets with reduced defect rate, cost and in less time [5], [6].

Current crowdsourced software development practices usually use an open call format such as online competitions. Although this enables wide task accessibility and self-selection features for the crowd developers, potential developers must choose from a dauntingly large set of task options. Chilton et al. found that [7] most workers usually view only a few recent tasks posted on the AMT micro-task crowdsourcing platform. Considering the various expertise and skill levels of the crowd developers, inappropriate developer-task matching may harm the quality of the software deliverables. Specifically, the caused issues can be viewed from two different perspectives:

*1) **From the developer's perspective**:* Developers from around the world can select the types of tasks in which they would like to compete, however under the situation that there are lots of simultaneously competitive tasks posted on crowdsourcing platforms such as TopCoder. Examining from a large number of tasks' descriptions, it is quite a laborious and demanding work for the developers to choose which one is more suitable for them to undertake. Usually there are more than one hundred simultaneously active online tasks on TopCoder for a single day. For instance, the number is 222 on Dec. 12, 2014, as shown in Figure 1.

*2) **From the platform's perspective**:* It is a valuable but challenging task for the platform to seek best available developers: a key factor for delivering qualified software assets. In traditional outsourced software development, one of the most critical steps is vendor selection, which has a direct correlation with the quality of the outcome [8]. In crowdsourced software development, despite the difference that the vendor developers are attracted by an open call format rather than being selected, encouraging the "right developers" to participate still plays a crucial role in delivering qualified clients' requested assets. Background and skill levels can vary significantly among crowd developers. Also, previous research showed that developer participation levels can impact CSD software quality [9]. Thus the platform may need to attract not only the "right developers", but also as many participants as possible.

[1]TopCoder Website: http://www.topcoder.com/

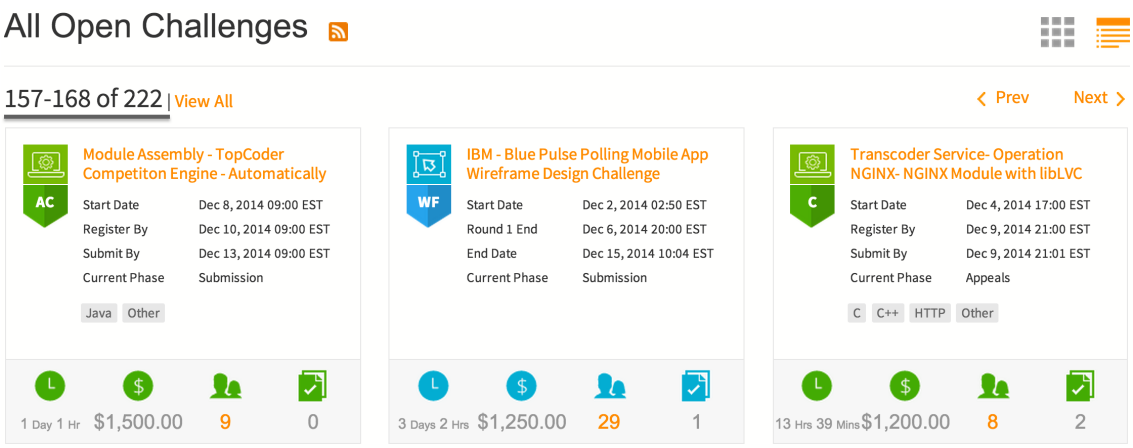

Fig. 1. Available tasks listed on TopCoder on Dec. 12, 2014.

In this paper, in order to tackle the above challenges, we employ content-based recommendation techniques to automatically match tasks to developers based on historical data and preference learning. The illustration of developer recommendation in the CSD context can be briefly shown in Figure 2. Although current CSD platforms such as TopCoder usually allow sign up for alerts about upcoming tasks [6], they do not provide personalised recommendation information for such alerts. Our developer recommendation results for newly arriving tasks can be delivered to signed up developers by email notification[2].

The primary contribution of this study is that we introduce a framework named *CrowdRex* which automatically recommends developers for newly arriving CSD tasks for different purposes: our system learns from the winning history for recommending reliable developers and learns from registration history to suggest suitable participants for available tasks. Other, more minor, contributions of the paper include multiple content features and their similarity measures as well as the formulation of the accuracy-diversity dilemma, for the CSD developer recommendation problem.

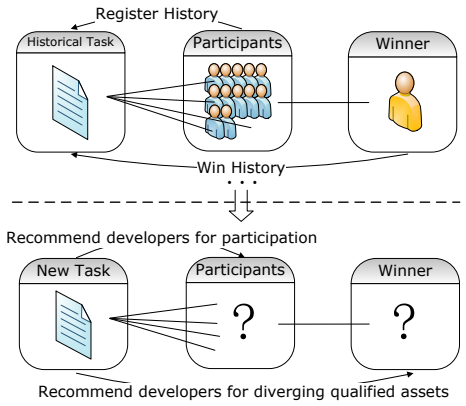

Fig. 2. Developer recommendation utilising historical data.

[2]According to a survey on TopCoder website (http://community.topcoder.com/tc?module=SurveyResults&sid=5851), 80.8% vote for "email" as their preferred method to receive information about TopCoder events.

The rest of this paper is organised as follows: Section II describes crowdsourced software development process, the work flow of open call tasks and participation incentives, as well as related recommendation techniques as background information. Section III introduces our approach to recommend developers for crowdsourced software development tasks. The evaluation of our work is presented in Section IV. Section V presents related work. Finally Section VI concludes and presents directions for future work.

II. Background

Prior to presenting our approach of developer recommendation, we outline the background relating to TopCoder's crowdsourced development process, open call tasks and related recommendation techniques for better understanding the construction of our recommender system. Other platforms may adopt similar processes but our evaluation was carried out using the historical data from TopCoder.

A. Crowdsourced Software Development

Since TopCoder is currently the world's largest CSD platform supporting a global crowd of more than 715,000 developers. It is based on this platform shall we introduce the typical CSD methodology.

*1) **Development Process**:* The crowdsourced software development process used by TopCoder generally follows a waterfall model, which is shown in Figure 3(a). Each of the phases is achieved by the open competition format, utilising TopCoder's global talent pool of developers.

The development process commences with a requirement phase. During this phase, the project manager, who comes from the crowd or the TopCoder platform, is responsible for managing the following phases and communicating with the client companies to identify their project goals, task plan and estimated budget. Then the requirements specification is defined and passed as the input to the next phase. The subsequent architecture phase decomposes the application into

a set of components. (TopCoder's crowdsourced software development practices component-based software engineering). The component design activity produces a full set of design documentations such as UML diagrams and component specifications.

These specified design components are then implemented in the subsequent development phase. The component development activity may incorporate pre-built reusable components. The finished components are combined together in an assembly phase and are further certified by system-level testing activities. Assembly tasks require online developers to build the application by assembling winning components according to the architecture design. Finally, the fully functioning solution is deployed into the customer's quality assurance environment in the deployment phase. After a period of user acceptance testing, all developed assets are delivered to the client. For further maintenance activities, TopCoder provides "Bug Hunt" and "Bug Race" tasks for discovering and fixing bugs.

*2) **Open Call Tasks**:* Crowdsourcing utilises an open call format to attract online workers to make contributions to the posted tasks. TopCoder's open call format is in the form of a competition. Each crowdsourced development task is organised as an open contest. Every registered member satisfying the legal requirement can register for contests and submit their solutions. Usually the top two winners receive prize money as a reward.

The typical open call task phases on TopCoder are divided into a series of task competitions. The duration of the whole process for a single task is usually 1-2 weeks. The process to run an open call task is illustrated in 3(b). To start with, a task categorised by its development type is posted on the website with information such as task descriptions, payment amount and time lines. The time lines include two important dates: registration deadline and submission deadline. All developers who are willing to participate should announce their decision publicly by registering the contest. This means online developers are allow to observe opponent developers' information including historical performance and skill ratings. This registration phase usually lasts a few days. Registrants can obtain detailed documentation and are required to submit their solutions before the submission deadline.

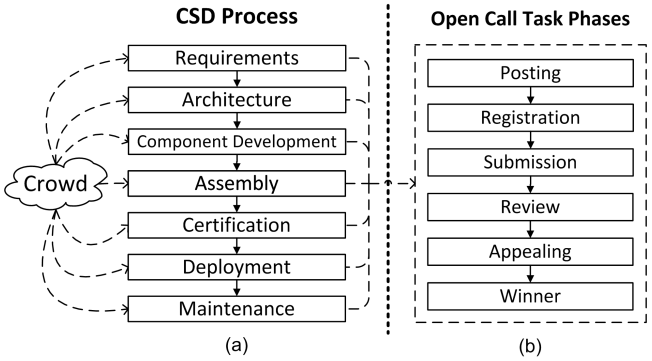

Fig. 3. Crowdsourced software development process and its task phases (derived from TopCoder.com and Mao et al. [10]).

After the submission deadline, all submitted solutions are collected by the platform to be evaluated by peer review according pre-defined screening and review scorecards[3]. The solutions that passed screening are scored (usually by three community experts from different perspective, e.g., performance on accuracy, stress and failure tests). Once the review process is finished, the developers are notified privately. If they are not satisfied with the results they have one chance to argue with the reviewers (called the appeal phase). Finally the revised scores are announced on the website and contestants are ranked by the average score given by all reviewers. The 1st place winner gets the full payment and the runner-up gets half of this amount.

*3) **Participation Incentives**:* The prize money is an important factor in motivating the crowd participation [11]–[13]. But not everyone register the competition for winning the prize. We respect the knowledge learned from register history. Statistics on TopCoder's historical assembly tasks shows that only 13.6% developers had ever won a task, 23.4% developers had not ever won a task but registered at least 5 tasks and 21.8% developers had not ever won a task but kept active in participation for at least 180 days. There are multiple types of incentives for developers to undertake CSD tasks. According to TopCoder, these incentives include gaining skills, getting feedback, making friends, earning money having fun, getting peer recognition and getting sense of accomplishment [14].

Since usually only top two winners of the tasks can be rewarded with money, the developers often get nothing monetary for their effort. But through competing in the task competition, developers may find it exciting and their submissions are tested by screening and further reviewed by the experts of the community. In this way they are rewarded with fun, acknowledgement and experts' feedback for improving their skills. According to a survey[4], 69.28% developers on TopCoder deem the platform has played at least a moderate role in improving their programming ability. What's more, TopCoder would announce "Coder of the Month" and top ten developers of each types of tasks on their website to satisfy developers' desire for recognition. These non-monetary incentives can be even more important than the money reward when motivating crowd developers for participation.

B. Recommendation Systems for Software Development

Recommendation systems for software development support developers who need to make decisions and choose from among a potentially overwhelmingly large set of possibilities. Examples of recommendation systems supporting software development include code navigation support, software change guidance, developer-bug assignment. An overview of available recommendation systems for software engineering can be found in the work of Robillard et al. [15].

[3]The scorecards are available at:
http://apps.topcoder.com/wiki/display/tc/Competition+Scorecards

[4]The online survey can be accessed at:
http://community.topcoder.com/tc?module=SurveyResults&sid=4765

Recommendation systems often employ either content-based or collaborative techniques (or use hybrids) [16]. In this paper we focus on content-based methods for developer recommendation. The fundamental assumption of content-base recommendation methods is that the user will be interested in the items that are "similar" (in some measurable sense) to the items the user preferred in the past. In the context of developer recommendation, in order to recommend developers for a newly arriving task, the content-based recommender tries to understand the commonalities among the tasks that the developers had participated. Then those developers whose commonalities are best matched with the newly arriving task would be recommended.

III. The Approach to Developer Recommendation for CSD Tasks

Our recommender system recommends developers in order to satisfy two objectives. One target is to recommend reliable developers for delivering qualified software assets. The other target is to recommend suitable developers who may be interested in registration for promoting the task participation level, and meanwhile reducing their effort in task selection.

A. *Our Proposed Developer Recommendation Framework*

The framework of our approach is named *CrowdRex*, which is shown in Figure 4. Our approach is based on content: a group of features describing a CSD task, such as required techniques, payment, title and overview description, post data and submission deadline, etc. The machine learner in the framework tries to capture the characteristics (e.g., expertise, expected payment and task duration, etc.) of each developer from their historical activities and recommend the best matched developers to newly arriving tasks. In order to satisfy the two targets described above, our approach is designed to extract empirical knowledge from developers winning and participation history separately to suggest two lists of developers corresponding to the objectives. The final list of recommended developers can adopt the two lists directly or use a re-rank method (e.g., prioritise developers according to their recent performance) to select a subset of developers according to the application context. In this paper we adopt the two lists of developers as our recommendation results.

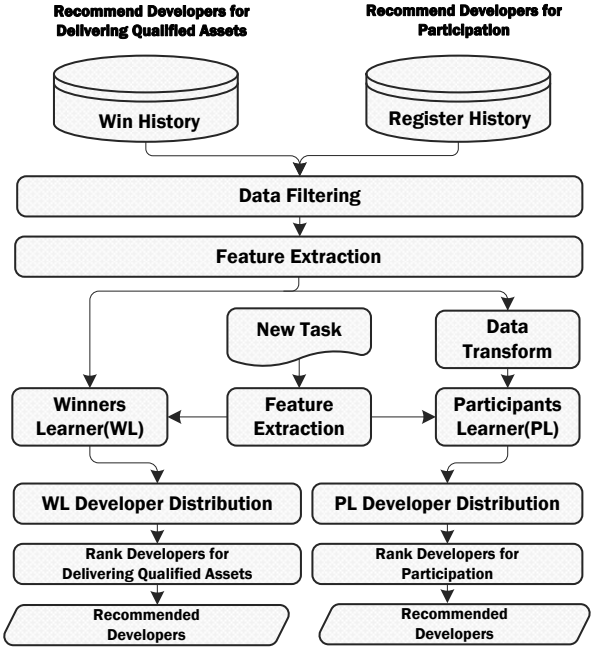

Fig. 4. CrowdRex: our developer recommendation framework.

B. *Recommend Developers for Delivering Qualified Assets*

In order to help the platform find reliable developers, we recommend skilled developers to undertake their familiar tasks. Our assumption here is that skilled developers undertaking familiar tasks are more likely to deliver qualified software assets.

To identify a qualified software asset, TopCoder has a review system (introduced in Section II) to guarantee software quality, which requires a qualified asset to pass a minimum review score. Generally, for the historical tasks, if there exists a winner, that means a qualified asset can be delivered. In this paper, we define a developer as "skilled" if he or she has delivered at least 5 qualified assets.

We employ multi-class, single-label classification technique to automate the process of identifying developers who may be familiar with the newly arriving tasks. Here we treat developer as class. Each task in historical records is labelled with the first place winner. The recommendation approach consists of four major phases:

*1) **Data Filtering**:* We filter those historical tasks (i.e. training set) with incomplete information (e.g., missing winner label or lack of task descriptions). To exclude irrelevance empirical knowledge, we remove the tasks which development type do not match current application domain (e.g., development, assembly, etc.). Also, all "non-skilled" developers are removed by the filter.

*2) **Feature Extraction**:* CSD tasks on TopCoder contain a variety of information. For feature extraction in this paper we consider the informative text features including title, description, programming language and techniques. Numeric features include task post date, allocated task duration and payment. The detailed feature description is presented in Table I. In order to unify numeric and text features, we convert each text feature into word vector format, keeping only meaningful and descriptive tokens processed by tokenization and stop words (e.g., 'a', 'the', 'and', 'of', 'is', 'this', etc.) removal. More formally, suppose there are m terms after the process for a text feature, the corresponding vector of this feature in task $t \in T$ would be $v_t = (w_{t,1}, w_{t,2}, ..., w_{t,m})$, where $w_{i,j}$ stands for the weight for each term $term_j$, which is calculated by Term Frequency-Inverse Document Frequency (TF-IDF) according to Equation 1:

$$w_{i,j} = \big(1 + log(tf_j)\big) * log\frac{|T|}{df_i} \quad (1)$$

TABLE I
THE CONTENT FEATURES FOR CROWDSOURCED SOFTWARE TASKS

Feature	Format	Description
Date	Numeric	Post date of the task.
PL	Text	What Programming Language is used.
Title	Text	Title of the posted task.
Tech	Text	Indicate what techniques are used.
Description	Text	Task descriptions overview.
Duration	Numeric	Time allocated to the task.
Payment	Numeric	How much US dollars will the winner get.

Where tf_j is the number of times $term_j$ appears in task i, and $|T|$ is the total number of tasks and df_i equals the number of tasks containing $term_j$.

*3) **Learner Training**:* In this step we train a classification model on prepared historical tasks. Various supervised learning algorithms can be adopted in this step, to generate a distribution of probabilities for all labels. Such algorithms can be C4.5 Decision Tree [17], NaïveBayes [18], k-Nearest Neighbour [19], etc.

*4) **Learner Applying**:* Lastly, we apply the trained model on test set, i.e. newly arriving tasks. Top N ranked winners from the generated probability distribution are recommended.

Note that the k-Nearest Neighbour algorithm mentioned in phase 3) is slightly different from other supervised learners because it does not have an explicit model training phase. However the lazy nature of this method makes it easier to understand and previous research have shown its efficiency and effectiveness. In order to use the k-Nearest Neighbour algorithm, we need to define the similarity measure. In this paper we define the similarity Sim between two task t_i and t_j according to the post date distance, matching in task programming language, matched number of techniques, allocated duration distance, payment difference and the text matching degree (i.e. cosine similarity of two vectors) in title and description. Sim is defined as Equation 2.

$$\begin{aligned} Sim(t_i, t_j) = w_1 Dis(F_{1,i}, F_{1,j}) + w_2 Dis(F_{2,i}, F_{2,j}) \\ + w_3 Dis(F_{3,i}, F_{3,j}) + ... + w_n Dis(F_{n,i}, F_{n,j}) \end{aligned} \quad (2)$$

Where w stands for the weight assigned to the corresponding feature (in this paper we use equal weights i.e. 1.0 for all features), Dis indicates the distance function which varies among different features. The definition of distance measures in this paper is shown in Table II.

C. Recommend Developers for Participation

By recommending developers for participation, we aim to enhance task participation level by helping developers to receive information on new arrival tasks which are suitable for them, meanwhile this can save their effort in task selection. Admittedly, compare with those reliable developers in the "winner group", these "long-tail" participants have much less chance to win the task prize. However their demand for participation still exists thus the recommendation service can be useful.

We define registrants of each historical task to be its "participants" in this paper. As there are multiple participants who register the newly arriving task to show their willingness to engage with the work. We treat developer recommendation for participation as a multi-class, multi-label classification problem. Previous research have suggested lots of methods to solve this problem. Tsoumakas and Katakis [20] grouped existing methods into two categories: problem transformation methods and algorithm adaptation methods. In this paper we use a straightforward data transform based method which fall into the first category. The method works as follows (the data filtering and feature extraction processes are the same as described above, which are omitted here):

Firstly, for each training historical task record with participants set P, decomposes the record into $|P|$ records, each of these records keeps the task features but corresponds to only one distinct participant $p \in P$. We apply a single-label machine learner that can generate a distribution of probabilities for all labels. Finally use the trained model to process the testing records, rank the participant labels according to the generated probability distribution and recommend top N developers.

IV. EMPIRICAL STUDY

This section presents the empirical study conducted on 2 datasets with different development characteristics to evaluate our proposed developer recommendation approach for CSD tasks. Since there is no existing study that evaluated any approach for CSD recommendation, we employ a simple but actionable method named "Active" as our baseline, which recommends the top N developers who have won most tasks for delivering qualified assets, and recommends the top N developers who have registered most tasks for participation, according to the given training dataset. Through the evaluation, we empirically study the following 3 research questions:

TABLE II
THE ADOPTED FEATURE DISTANCE MEASURES

Feature	Distance Measure
Date	$(Date_i - Date_j)/Date_{MaxDiff}$
PL	$PL_i == PL_j \ ? \ 1 : 0$
Title	$\frac{Tit_x \cdot Tit_y}{\|Tit_x\| \|Tit_y\|}$
Tech	$Match(Tech_i, Tech_j)/NumberOfTechs_{Max}$
Description	$\frac{Des_x \cdot Des_y}{\|Des_x\| \|Des_y\|}$
Duration	$(Duration_i - Duration_j)/Duration_{Max}$
Payment	$(Payment_i - Payment_j)/Payment_{Max}$

TABLE III
STATISTICS OF THE EVALUATION DATASETS

Dataset	# Tasks	# Reg.	# Win.	Duration
Development	1093/1367	1045/3533	92/298	2003.10-2013.02
Assembly	1505/1727	541/1547	86/211	2008.11-2013.03

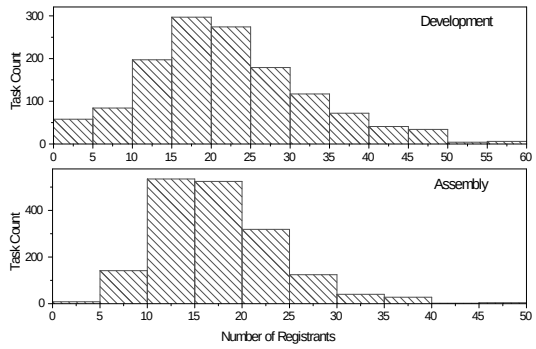

Fig. 5. The distribution on the number of registrants in each task.

RQ1. (Baseline Comparison): How does our proposed recommendation approach (instantiated with different machine learners) perform compare to the "Active" baseline method? Since the baseline approach is simple and actionable, if we cannot outperform this method then there is no reason for the platform to adopt our recommender system.

RQ2. (Performance Assessment): Which of the machine learners can best support our recommender system according to accuracy and diversity metrics? Can the best learner consistently outperform other learners, on different datasets and for different recommendation purposes?

RQ3. (Insights): What useful insights from the performance data can we yield for the real world CSD platform?

A. *Dataset*

We evaluate our approach on the novel datasets collected from TopCoder, which currently has the largest community for crowdsourced software development. We collected 3,094 historical tasks that have been crowdsourced between Oct. 2003 and Mar. 2013, covering 2 types of software development tasks (component development and assembly). We evaluate our approach separately on each of the 2 datasets. Summary data regarding the 2 datasets after and before data filtering is shown in Table III. Note that the 3rd and 4th columns indicate the number of distinct registrants and winners respectively. Figure 5 presents the registrations distribution among the 2 types of CSD tasks. The statistics generally indicates a normal distribution for the number of registrants in each task.

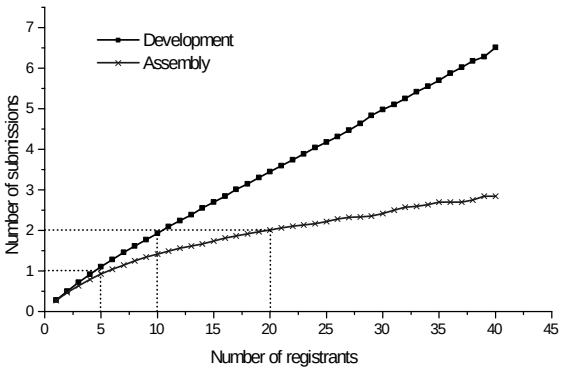

Fig. 6. The relationship between the number of registrants and submissions.

B. *Study Setting*

We sort the task records ascendingly according to their posted time and then divide each of the 2 datasets into 10 folds. The top 9 folds of records are selected as our training set and the 10th fold is selected as our testing set, i.e. newly arriving tasks. On each of these datasets, we recommend top 5, 10 and 20 developers separately and record the recommended lists for calculating evaluation metrics. Note that here we choose to recommend 5, 10 and 20 developers, which produce larger sets of candidate developers compare to most studies on bug triage. This is because in the CSD context, a wider range of developers are encouraged to participate, and the parameters of top 5, 10 and 20 developers basically matches the real registrants distribution shown in Figure 5: For component development tasks, the average registrants number is 16, and the 10th-90th percentile corresponds to 10-25 registrants. For assembly tasks, there are 20 registrants on average for a task and the 10th-90th percentile corresponds to 9-36 registrants. Besides, Figure 6 presents the relationship between the number of registrants and number of qualified submissions. We can see that at least 5 registrants are expected in order to receive one submission for a task. Also, 10 and 20 registrants are expected to obtain 2 submissions for the development and assembly tasks separately. We choose to recommend 5, 10 and 20 developers for simulating the real application of our approach.

The machine learning algorithms evaluated in our study are selected from popular algorithms applied in previous related research, which include C4.5 Decision Tree (C4.5), NaïveBayes (NB) and k-Nearest Neighbour when $k = 1$ (KNN_1) and $k = 5$ (KNN_5). For these machine learners applied in the experiments, we do not optimise their parameters for a certain method to avoid biasing. Our results are therefore directly compared to the baseline, which also requires no tuning. The corresponding parameters are set according to the default parameters in the popular open sourced data mining tool named Weka [21].

C. *Evaluation Metrics*

In order to evaluate the performance of our proposed approach, we need metrics that can be used to assess developer recommendation techniques. Previous related research topics such as bug triage focus on the *Accuracy* dimension of the performance, the typical metrics used are *Precision* and *Recall*. However in the context of CSD tasks, we additionally need to

TABLE IV
ACCURACY AND DIVERSITY OF DEVELOPER RECOMMENDATION FOR DELIVERING QUALIFIED SOFTWARE ASSETS

Dataset	Recommend	C4.5		NaïveBayes		KNN_1		KNN_5		Active	
		Acc	Div	Acc	Div	Acc	Div	Acc	Div	Acc	Div
DEV	5	**50%**	**40%**	44%	21%	34%	27%	32%	35%	37%	5%
	10	**60%**	**45%**	54%	26%	44%	30%	49%	38%	46%	11%
	20	**71%**	**52%**	67%	18%	56%	40%	61%	42%	57%	22%
ASM	5	37%	72%	33%	33%	38%	47%	**43%**	**73%**	15%	6%
	10	51%	74%	44%	18%	49%	49%	**58%**	**76%**	26%	12%
	20	63%	77%	54%	48%	57%	53%	**67%**	**78%**	37%	23%

TABLE V
ACCURACY AND DIVERSITY OF DEVELOPER RECOMMENDATION FOR PARTICIPATION

Dataset	Recommend	C4.5		NaïveBayes		KNN_1		KNN_5		Active	
		Acc	Div	Acc	Div	Acc	Div	Acc	Div	Acc	Div
DEV	5	**30%**	3%	4%	**15%**	3%	8%	3%	7%	24%	1%
	10	**23%**	6%	2%	**17%**	5%	14%	6%	13%	18%	1%
	20	**21%**	11%	1%	18%	6%	**20%**	6%	19%	12%	2%
ASM	5	**69%**	1%	12%	10%	10%	**35%**	10%	34%	65%	1%
	10	**48%**	2%	6%	11%	16%	**47%**	17%	46%	44%	2%
	20	**30%**	4%	3%	12%	17%	**53%**	18%	53%	25%	4%

consider *Diversity*, which aims to encourage more developers to migrate into the task as the essence of crowdsourcing is to utilise the "wisdom of the crowd". A recommender system can be statistically accurate if we focus on optimising the *Accuracy* dimension but it may be not very useful for practical purpose if it has poor diversity. For example, suppose a developer d participates in all the CSD tasks and the developer recommender would recommend d to any newly arriving tasks in order to be accurate. However since d is so active, he or she would likely still participate the newly arriving tasks without being recommended. If the recommender system focus only on accuracy, it will fail to enhance any participation level.

For measuring the *Diversity* dimension, we follow Adomavicius and Kwon [22] referred diversity to the number of distinct developers that can ever be recommended by the recommender system. To normalise the diversity measure, we define it as the percentage format. This diversity metric can be calculated according to Equation 3 where $R_i\,(t))$ stands for the recommended i developers for task t, $Actual\,(t)$ stands for the ground truth developers for task t, Tr stands for training set and T stands for testing set. As for the *Accuracy* metrics, we define $HitRate_i$ to assess the recommender for delivering qualified assets (one ground truth label for each task) and use $AveragePrecision$ to assess the recommender for participation (multiple ground truth labels for each task), i.e. the *Accuracy* dimension in Table IV is measured from $HitRate$ and the *Accuracy* dimension in Table V is measured from $AveragePrecision$. These two metrics are defined formally as shown in Equation 4 and Equation 5.

$$Div_i = \left|\bigcup_{t\in T} R_i\,(t)\right| \Big/ \left|\bigcup_{t\in T_r} Actual\,(t)\right| \tag{3}$$

$$HitRate_i = \frac{1}{|T|} * \sum_{t\in T} \left|correct\big(R_i\,(t)\big)\right| \tag{4}$$

$$AvgPrec_i = \frac{1}{|T|} * \sum_{t\in T} \left(\left|correct\big(R_i\,(t)\big)\right| / \left|R_i\,(t)\right|\right) \tag{5}$$

D. *Results and Analysis*

Table IV shows the results for delivering qualified assets evaluated on the datasets of Component Development (DEV) and Assembly (ASM) CSD tasks. Table V presents the results for participant recommendation on these two datasets. A performance comparison of the results in above tables is illustrated by the charts in Figure 7.

*1) **Results for RQ1 (Baseline Comparison)**:* The performance of our recommender system outperforms the baseline "Active" method across all datasets with different settings (recommending 5, 10, 20 developers). When recommending developers for delivering qualified assets, our recommender system instantiated with any of the 4 machine learners performs better than the baseline method for most cases, assessing from *Accuracy* and *Diversity*. When recommending developers

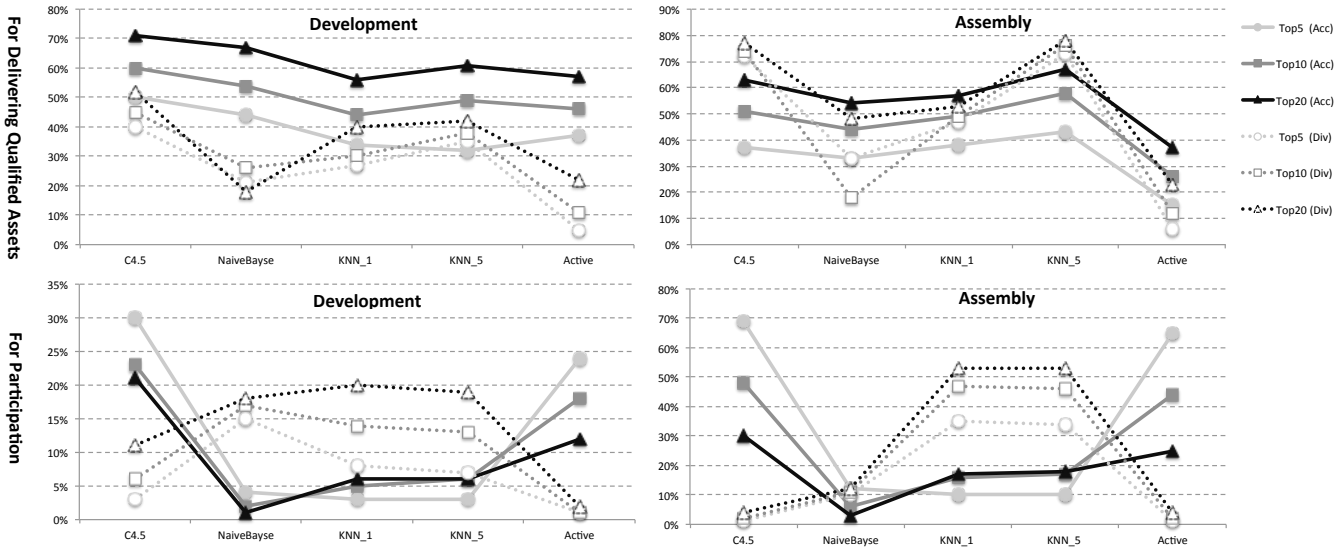

Fig. 7. Performance comparison of developer recommendation when recommending 5, 10 and 20 developers. The x-axis shows the machine learners and the baseline method "Active". The y-axis shows the value for *Accuracy* (Acc) and *Diversity* (Div) measures. The scatter points are linked for the purpose of improving the readability.

for participation, since the baseline method shows good capability to recommend developers accurately, the best performed learner C4.5, on average is only 5.5% better than baseline from the *Accuracy* perspective, and from the *Diversity* perspective, our recommender employ different machine learner shows different capabilities, i.e. the KNN_1 (8%-53%), KNN_5 (7%-53%) and NaïveBayes (10%-18%) learners performs much better than the baseline (1%-4%) method.

*2) **Results for RQ2 (Performance Assessment)**:* As shown in Table IV and Table V, there does not exist a machine learner that can consistently support our recommender system better than other machine learners. However the C4.5 decision tree learner performs best for *Accuracy* in 9 of all 12 cases which indicates that it is the best learner for *Accuracy*. Evaluating from the *Diversity* dimension, KNN_1 is the best learner which outperforms others in 4 of 12 cases. When we assess from both *Accuracy* and *Diversity* dimensions by calculating their Harmonic mean (i.e. $2 * Accuracy * Diversity/(Accuracy + Diversity)$), C4.5 decision tree outperforms others in 5 of 12 cases. Thus we regard C4.5 as the best learner for our proposed recommender system, when assessing from the Harmonic mean of *Accuracy* and *Diversity* measures.

From Figure 7 we can see that when recommending developers for delivering qualified assets, our proposed recommender system can achieve reasonable accuracy and diversity. However we cannot reach both high accuracy and diversity when recommending developers for participation. This reveals an *Accuracy-Diversity* dilemma of our recommender system.

Observed from the performance of the baseline "Active" method, when recommending developers for participation, the biased learner with high accuracy and low diversity can be attributed to the ground truth of the participants distribution, i.e., there does exist the phenomenon that a certain number of active developers tend to participate newly arriving tasks persistently. Admittedly, the recommender system with high accuracy and low diversity consistently recommend those active developers can be a solution, but this only works for short-term benefit. Only by achieving reasonable diversity, can the recommender serve for the "long-tail" developers and enhance the task participation level (i.e., exploration). Meanwhile, the recommendation can recommend some of the active developers to ensure the accuracy is acceptable (i.e., exploitation). We suggest that a proper trade-off between exploration and exploitation may help the recommender system work for long-term benefit.

*3) **Results for RQ3 (Insight)**:* Our empirical results can be used to derive a few insights which are listed as follows:

1). In conclusion, from the performance data in Table IV and Table V, we suggest some careful selection from among different machine learners will be required according the application context (e.g., task type). This conclusion is consistent with the no-free-lunch theorem [23] for machine learning.

2). Those most active developers are very likely to participate in newly arriving tasks. However they are only a small proportion of all developers, shown from the high accuracy and low diversity of "Active" method. This indicates that, to enhance the participation level, the CSD platform may need to allocate higher weight to diversity to focus on recommending suitable, but not necessarily the most active developers. The winners of newly arriving tasks tend to be more various according to the high diversity in Table V, which may indicate that skilled developers have different knowledge background and tend to deliver qualified assets for those matched tasks within their expertise.

3). The baseline "Active" method is simple and actionable,

but can be an option to recommend developers for participation if the low diversity can be tolerated. It may yield short-term advantage at the expose of the long-term sustainability of the CSD platform.

V. Related Work

Crowdsourced Software Development (CSD) has received increasing attention from the research community in recent years. The research topics vary from CSD model to domain application. Kazman and Chen [24] argued that traditional software development life-cycle models such as the waterfall model and the spiral model are inadequate for mass peer production and the service nature of crowdsourcing. They proposed the Metropolis model for the development of crowdsourced systems. For the practice of CSD, Prikladnicki et al. [25] reported their starting point of a multi-year study on CSD in Brazil, which aims to identify the challenges as well as the best practices for CSD. Musson et al. [26] showed how they utilised the crowd users to improve the Lync software performance at Microsoft. Crowdsouring can also be used to tackle classical problems in software engineering. Schiller and Ernst [27] presented VeriWeb, which is an online IDE for solving the skill barriers problem in verification. Their experiments showed that VeriWeb can lower the monetary and time cost of verification. However the workers need to be professional contract workers rather than ad-hoc workers. Pastore et al. [28] conducted a study on using the crowd to solve the oracle problem [29] in software testing. Their experimental results indicated that although it still remains hard to obtain qualified solutions from the crowd, it can be a viable method to ameliorate the oracle problem. For the quality concern of CSD outcomes, Li et al. [9] analysed the key factors on CSD software quality. By conducting an empirical study on TopCoder.com, they suggested four critical factors in improving the quality of the deliverables, including the prosper level of the CSD platform, the size of the task, the maximum skill level of the crowd developers as well as the software design quality of the CSD task.

Our review of the literature reveals that few previous authors have investigated recommendation-related topics for the emerging crowdsourcing paradigms. We could not find any peer-reviewed papers on developer recommendation in the CSD context. However, for the more general context of recommending crowdsourced micro-tasks to users, we did find related work although this is not directly related to CSD recommender systems. Ambati, Vogel and Carbonell [30] proposed a recommendation approach based on implicit modelling of interest and skills. Yuen, King and Leung [31] employed probabilistic matrix factorization for preference-based task recommendation in crowdsourcing system. These two studies were conducted on the Amazon Mechanical Turk platform with micro-tasks, rather than tasks as complex as software development.

Another related topic which has been studied more extensively is bug triage. Čubranić and Murphy [32] proposed to automate bug triage using text categorization techniques. Anvik, Hiew and Murphy [33] expanded the previous work by data pre-processing, using additional features and exploring the performance of more machine learning algorithms. Gaeul, Kim and Thomas [34] further improved bug triage accuracy using developers tossing behaviours in bug repositories. Matter, Kuhn and Niestrasz [35] presented their approach to automatically suggest developers for handling bug reports using a vocabulary-based expertise model. Tamrawi et al. [36] introduced fuzzy set and cache-based modelling techniques for automatic bug triage. Xuan et al. [37] leveraged developer prioritization based on a social network techniques to assist triage tasks in bug repositories. Wang et al. [38] explored developers' collaboration relationship in bug repositories. They model the collaboration as heterogeneous graphs to help triage the bug reports and their evaluation showed higher accuracies can be achieved by utilising the heterogeneous graphs. However these proposed approaches need to be carefully re-examined in the CSD context, due to the specialized properties and peculiarities of CSD which require special treatment. For instance, one prior research has shown that traditional laws on software cost are challenged in the CSD context [10]. Likewise, we consider traditional bug triage methods may not migrate to this new context directly because previous bug triage studies are based on open source communities, which are quite different from crowdsourcing community.

VI. Summary

Crowdsourced software development is an emerging paradigm which utilises the "wisdom of the crowd" for software production. The CSD tasks demand reliable developers and enough participation to guarantee a qualified asset that can be delivered to the client. In addition, examining from hundreds of tasks' descriptions, it is quite a laborious and demanding work for the developers to choose which one is more suitable for them to participate.

To tackle above challenges, we introduced content-based recommendation techniques for developer recommendation in the emerging CSD context. Our system learns particular interests from registration history and mines winner history to favour appropriate developers. To encourage task participation as well as to serve more developers, we focus not only accuracy but also diversity to measure the performance of our recommender system. The experimental results show that our recommender system outperforms the baseline method and can achieve promising accuracy and diversity. We also provide a few insights concluded from the results for the CSD platform and its online developers.

For future work we plan to propose an improved approach to optimise recommendation results by considering developers' strategic behaviours and their social network information. A wider range of machine learners and related state-of-the-art methods in bug triage would be evaluated for performance comparison. In addition, to further investigate the usefulness of the recommender system, an online empirical study would be conducted.

ACKNOWLEDGMENT

The research is funded by UCL Graduate Research Scholarships (GRS), UCL Overseas Research Scholarships (ORS) and Dynamic Adaptive Automated Software Engineering (DAASE) programme grant (EP/J017515). The authors would like to thank TopCoder for giving us the permission to use their CSD task data for this work.

REFERENCES

[1] A. Begel, J. D. Herbsleb, and M.-A. D. Storey, "The future of collaborative software development." in *CSCW (Companion)*, S. E. Poltrock, C. Simone, J. Grudin, G. Mark, and J. Riedl, Eds. ACM, 2012, pp. 17–18.

[2] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.

[3] Massolution, "Crowdsourcing industry report," 2012.

[4] N. Archak, "Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on topcoder.com." in *WWW*, M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, Eds. ACM, 2010, pp. 21–30.

[5] K. Lakhani, D. Garvin, and E. Lonstein, "Topcoder (a): Developing software through crowdsourcing," *Harvard Business School Case*, no. 610-032, 2010.

[6] A. Begel, J. Bosch, and M.-A. D. Storey, "Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder." *IEEE Software*, vol. 30, no. 1, pp. 52–66, 2013.

[7] L. B. Chilton, J. J. Horton, R. C. Miller, and S. Azenkot, "Task search in a human computation market," in *Proceedings of the ACM SIGKDD Workshop on Human Computation*, ser. HCOMP '10. New York, NY, USA: ACM, 2010, pp. 1–9.

[8] V. Wadhwa and A. R. Ravindran, "Vendor selection in outsourcing," *Computers & Operations Research*, vol. 34, no. 12, pp. 3725–3737, 2007.

[9] K. Li, J. Xiao, Y. Wang, and Q. Wang, "Analysis of the key factors for software quality in crowdsourcing development: An empirical study on topcoder.com." in *COMPSAC*. IEEE Computer Society, 2013.

[10] K. Mao, Y. Yang, M. Li, and M. Harman, "Pricing crowdsourcing-based software development tasks," in *Proceedings of the 2013 International Conference on Software Engineering, New Ideas and Emerging Results Track*, ser. ICSE 2013. IEEE Press, 2013, pp. 1205–1208.

[11] J. Antin and A. Shaw, "Social desirability bias and self-reports of motivation: A study of amazon mechanical turk in the us and india," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI'12. New York, NY, USA: ACM, 2012, pp. 2925–2934.

[12] N. Kaufmann, T. Schulze, and D. Veit, "More than fun and money. worker motivation in crowdsourcing: a study on mechanical turk," in *Proceedings of the Seventeenth Americas Conference on Information Systems*, 2011, pp. 1–11.

[13] M.-C. Yuen, I. King, and K.-S. Leung, "A survey of crowdsourcing systems," in *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, Oct 2011, pp. 766–773.

[14] Jim McKeown, "Open source meets capitalism: Collaborate by competing." [Online]. Available: http://opensource.com/business/12/6/open-source-meets-capitalism-collaborate-competing

[15] M. P. Robillard, R. J. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, vol. 27, no. 4, pp. 80–86, July/August 2010.

[16] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734 – 749, June 2005.

[17] J. R. Quinlan, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[18] P. Langley, W. Iba, and K. Thompson, "An analysis of bayesian classifiers," in *In Proceedings of the 10th International Conference on Artificial Intelligence*. MIT Press, 1992, pp. 223–228.

[19] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[20] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Int J Data Warehousing and Mining*, vol. 2007, pp. 1–13, 2007.

[21] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[22] G. Adomavicius and Y. Kwon, "Toward more diverse recommendations: Item re-ranking methods for recommender systems," in *Proceedings of the 19th Workshop on Information Technology and Systems (WITS'09)*, Dec. 2009.

[23] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 67–82, 1997.

[24] R. Kazman and H.-M. Chen, "The metropolis model a new logic for development of crowdsourced systems," *Commun. ACM*, vol. 52, no. 7, pp. 76–84, Jul. 2009.

[25] R. Prikladnicki, L. Machado, E. Carmel, and C. R. B. de Souza, "Brazil software crowdsourcing: A first step in a multi-year study," in *Proceedings of the 1st International Workshop on CrowdSourcing in Software Engineering*, ser. CSI-SE 2014. New York, NY, USA: ACM, 2014, pp. 1–4.

[26] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, and S. Ganguly, "Leveraging the crowd: How 48,000 users helped improve lync performance," *IEEE Softw.*, vol. 30, no. 4, pp. 38–45, Jul. 2013.

[27] T. W. Schiller and M. D. Ernst, "Reducing the barriers to writing verified specifications," in *Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2012)*, Tucson, AZ, USA, October 23-25, 2012, pp. 9–112.

[28] F. Pastore, L. Mariani, and G. Fraser, "Crowdoracles: Can the crowd solve the oracle problem?" in *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, ser. ICST '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 342–351.

[29] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, 2015, to appear.

[30] V. Ambati, S. Vogel, and J. G. Carbonell, "Towards task recommendation in micro-task markets." in *Human Computation*, ser. AAAI Workshops, vol. WS-11-11. AAAI, 2011.

[31] M.-C. Yuen, I. King, and K.-S. Leung, "Taskrec: probabilistic matrix factorization in task recommendation in crowdsourcing systems," in *Neural Information Processing*. Springer, 2012, pp. 516–525.

[32] D. ubrani, "Automatic bug triage using text categorization," in *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. KSI Press, 2004, pp. 92–97.

[33] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*. ACM, 2006, pp. 361–370.

[34] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on The Foundations of Software Engineering*, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 111–120.

[35] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, ser. MSR '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 131–140.

[36] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 365–375.

[37] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proceedings of the 2012 International Conference on Software Engineering*, ser. ICSE 2012. Piscataway, NJ, USA: IEEE Press, 2012, pp. 25–35.

[38] S. Wang, W. Zhang, Y. Yang, and Q. Wang, "Devnet: Exploring developer collaboration in heterogeneous networks of bug repositories," in *Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on*, Oct 2013, pp. 193–202.