Failure Prediction in Crowdsourced Software Development

Abdullah Khanfor, Ye Yang, Gregg Vesonder
Stevens Inst. of Technology
Hoboken, NJ USA
e-mail: {akhanfor, ye.yang, gvesonde}@stevens.edu

Guenther Ruhe
University of Calgary
Calgary, Canada
e-mail: ruhe@ucalgary.ca

Dave Messinger
TopCoder, Inc.
Glastonbury, CT USA
e-mail: messinger@topcoder.com

***Abstract*— Background: Despite the increasingly reported benefits of software crowdsourcing, one of the major practical concerns is the limited visibility and control over task progress. Aim: This paper reports an empirical study to develop a framework for failure prediction in software crowdsourcing. Method: This process begins with identifying 13 influencing factors in software crowdsourcing failures, across four categories including task characteristics, technology popularity, competition network, and workers reliability. Presenting an algorithm to construct worker competition network and extract its network metrics features. The proposed framework was evaluated on 4,872 software crowdsourcing tasks, extracted from TopCoder platform, using five machine learners, compared with in-house TopCoder predictor. Results: 1) Workers reliability, links in the description, number of registered workers, number of required technologies, and task-workers network modularity are the most influencing factors for predicting crowdsourcing failure; 2) The top-three learners for task failure are Naïve Bayes, Random Forest, and StackingC, with precision above 98.8%, recall above 81.2%, and F-measure above 91.2%; and 3) The proposed best learners significantly outperform the two baseline models in our evaluation. Conclusions: The performance of the proposed framework is better than those of the two baseline models. This paper offers practical recommendations for managing task failure risks.**

Keywords-software crowdsourcing, failure prediction, developer reliability, technology popularity, machine learning.

I. INTRODUCTION

Compared to in-house software development, the practice of crowdsourced software development (CSD) claimed the ability to deliver customer requested software assets with a lower defect rate, at lower cost, in less time [1] [2]. However, major practical concerns are largely associated with the limited control over task progress and deliverable quality.

In the context of CSD, projects are typically organized into dozens or hundreds of mini-tasks, each following the process flow of task preparation by a requesting company, task posting to a crowdsourcing platform, task registration and submission by crowd workers, and peer review. The whole process is depicted in Fig. 1. Scores from peer reviews primarily characterize the task completion status. For example, in the TopCoder platform [1] [2], a successful task submission must be scored at least 75 out of 100. Any submission with a review score less than 75 is considered as failure, and marked with a status of "Cancellation-Failed Review". If a task receives no submissions, it is labeled as "Cancellation – Zero Submissions".

In this study, we call these two scenarios "Task Cancellation" and "Task Starvation" respectively, and use the term "Task Failure" to represent these two types of unsuccessful CSD tasks. It's of our interest to study the factors associated with such failure and its predictability, because for task requesters, employing external, unknown, uncontrollable crowd workers places their projects under greater uncertainty and risk compared with in-house development. For example, one case study [3] presented the preliminary results of a multi-year study on crowdsourcing in the Brazilian IT industry, and reported interviews that highlighted the concerns about the crowdsourced software quality. Understanding the influencing factors of task failure becomes extremely important for managers when predicting and reacting to task failure. In a recent work, Dubey et al. investigated five influence factors for CSD task completion [12] and proposed task completion prediction methods. However, none of the factors takes into consideration the effects of task-worker competition network. Our study seeks to address these issues and focus on improving the predictability of task cancellation risks, by answering the following research questions:

- *RQ1: What are the influencing factors that are associated with CSD task failure and their relative importance to failure prediction?*
- *RQ2: Can we build predictive models for CSD task*

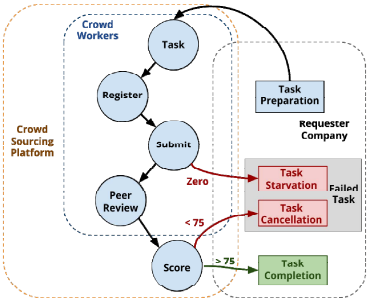

Fig. 1. A Typical Software Crowdsourcing Process.

978-1-5386-3681-7/17 $31.00 © 2017 IEEE
DOI 10.1109/APSEC.2017.56

failure based on the identified factors?

- *RQ3: To what extent does the proposed predictive models improve the current baselines?*

This paper reports our empirical studies towards investigating and deriving solutions to the research questions. We first present a motivating example, then propose a CSD failure prediction framework consisting of a list of attributes characterizing four categories including task characteristics, technology popularity, worker characteristics, and task-worker competition network. We then develop algorithms to support automatically extract and computation of such attributes, and build different failure predictive models using six different machine learning algorithms, evaluated on a dataset extracted from the TopCoder platform. Finally, we compare the model performance with two baselines. Evaluation results of the machine learners show that our proposed prediction methods outperform both baselines. The rest of the paper is organized as follows: Section II introduces a motivating example; Section III presents the failure prediction framework; Section IV introduces the design of empirical study; Section V reports the empirical results; Section VI Discussion of the results; Section VII is the review of related literature, and finally Section VIII is the conclusion.

II. A MOTIVATING EXAMPLE

Table I shows an example of 9 selected tasks within one project from the TopCoder website. The project contains a total of 19 tasks dated between March 12, 2014 and June 30, 2014. As seen in Table I, Task 1 was subsequently cancelled because there were zero submissions. The task was simplified and completed as Task 2, with 1 submission. However, Tasks 3 - 8 exemplify an important but rather frustrating case, Basically the same task was posted six times and the prize was increased twice (i.e. in Task 5 and 7) serving as an additional incentive, and the scope of the requirements was reduced in Task 8, but all end up with task cancelled nonetheless. For such a 5-day micro-task as originally planned in Task 3, the resultant cumulative schedule delay due to a series of cancellations is about 40 days (4/2/2014 - 5/12/2014). Considering the total duration of this project, the schedule overrun caused by task cancellation is as high as 40%.

Noted from the challenges names in Table I, six tasks (#3-#8) are with the same challenge name and all cancelled due to failure. Task #9, with a reduced scope, was successful. This indicates the potential failure factors associated with scope of task requirements. Task scope should be appropriated reflected into main motivating factor such as price, type and duration, since underpriced tasks tend to fail due to less appealing to workers. It is also noted that the links included in the task description contribute in additional documents for the workers to understand the task details or constraints. When comparing worker differences, in Task 5 through 7, the average worker reliability score is noted to be relatively low, between 0.07 to 0.13. In contrast, Task 9 have an average workers' reliability score of 0.16 and the highest reliability score of one worker is 0.6. Reliability score-related metrics is a good indicator for task outcome, as reported in [12].

Our further analysis shows that among a total of 4,872 tasks, there were 754 cancelled tasks. This cancellation rate corresponds to a failure rate of 15.4%, which is very high. This is consistent and encouraging results compared with the 60%-80% probability of success for competitions with at least one registrant reported in [30]. However, the 15.4% failure rate is non-trivial which necessitates analytically-based prediction models to help assess and manage failure risk as early as possible. In Dubey et al.'s study [12], there is a lack of in-depth feature analysis to provide early-stage prediction for task completion outcomes. Further, none of the factors takes into consideration the effects of task-worker competition network. Social network can be affected by social behaviors. Wu et al. compared two CSD projects competition networks and argued that denser task-worker interaction contributes to CSD success [14]. Mao et al. also proposed to recommend appropriate tasks to workers based on worker's social prestigious network [9]. Inspired by these studies, we consider including task-worker competition network metrics and explore its role in CSD task failure prediction. We aim at identifying further less intuitive yet more relevant factors that lead to potential failure.

III. PROPOSED FAILURE PREDICTION FRAMEWORK

In this study, task prediction is modelled as a binary classification problem, with successful tasks labelled as "Completed, 1", and failed tasks labelled as "Cancelled, 0". To address the task failure issue, we develop a failure prediction framework, as shown in Fig. 2. We start by establishing a set of metrics from the "TASK", "TECH",

TABLE I. EXAMPLES OF CANCELLED TASKS ON TOPCODER PLATFORM.

Task	Status	Challenge Name	Start Date	End Date	Prize	#Reg.	#Sub.
1	Cancelled	Urania Job Detail and Job Search Screens	3/29/2014	4/1/2014	1500	12	0
2	Completed	Urania Job Detail and Job Search Screens - Reduced Scope	4/1/2014	4/3/2014	600	5	1
3	Cancelled	Urania Job - Application Form	4/2/2014	4/6/2014	750	9	0
4	Cancelled	Urania Job - Application Form	4/7/2014	4/14/2014	700	8	0
5	Cancelled	Urania Job - Application Form	4/17/2014	4/22/2014	1100	8	0
6	Cancelled	Urania Job - Application Form	4/23/2014	5/23/2014	1100	22	0
7	Cancelled	Urania Job - Application Form	5/3/2014	5/9/2014	1500	0	0
8	Cancelled	Urania Job - Application Form	5/9/2014	5/12/2014	750	5	0
9	Completed	Urania Job - Application Form SCOPE REDUCED!!	5/12/2014	5/16/2014	1600	15	2

"NETWORK" and "WORKER" categories, with the intention to explore the relative significance of these factors in influencing task success or failure. Then employing machine learning based prediction techniques, we implement the framework for preprocessing data from the historical dataset, training learners and using the trained learners for predicting new tasks. Next, we will present the main elements in the framework, and the Evaluation component and three research questions (RQ1, RQ2, and RQ3) will be introduced in Section IV.

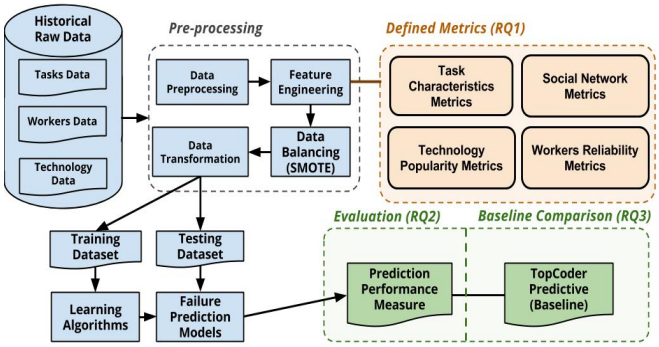

Fig. 2. Failure Prediction Framework.

A. Metric Definition

The four categories of attributes identified in this study include:

1) Task characteristics (TASK), which contains data on the award, type, duration, and link count as task size indicator;
2) Technical complexity and popularity (TECH), which contains data on both the number and popularity of platforms/technologies required to be used in a task;
3) Worker characteristics (WORKER), which includes the number of workers registered, and the average and maximum reliability of the registered workers;
4) Competition Network (NETWORK), which represents the results of graph properties calculated from Bipartite graph with workers and tasks as nodes, worker registration in task as directed edge, and modularity, PageRank and strongly connected components as the chosen network metrics.

Table II provides details of the metric descriptive statistics. Some metrics such as Award, type, duration are straightforward and used in existing studies. Next, we introduce the derived metrics proposed by our study in more detail.

1) Measuring Link Count as Task Size Indicator

Classical size measures such as lines of code are not accessible in our study through the dataset. Alternatively, we introduce an additional metrics to measure the size of task based on the number of external website links provided in the task description *link count* (measures the number of external website links contained in the task description). In TopCoder, a task description is basically natural language text, with largely varied formats and lengths. External links to dependent interfaces/modules or screen mockups are frequently provided as detailed requirement descriptions. A large chunk task could have a very long task description, or just contain a link to an external file with very complicated screen mockups. Therefore, we argue that link count could be a better indicator for task size than lines of code count in our study.

2) Deriving Technology Popularity

In crowdsourcing, technical complexity and popularity have a direct impact on the scope and availability of worker resources that a task is targeting. Typically, task description involves multiple technologies and platform requirements. We group similar technologies into categories and that result in 30 different technology categories. For example, JavaScript technologies are represented in multiple tags involved in a task description. The technologies are elaborated, in which "Node.js", "Angular.js", "JQuery", and "JavaScript" are different technologies in the raw dataset. We did group similar technologies as one technology and grouped other technologies in similar way which results in 30 different technologies. In addition, to measure the popularity of technologies, we give a weighted value for each task depending on the frequency of occurrence of technologies and platforms in the whole dataset. First, we extract and compute the number of technologies involved in a task from its "technologies" attributes. Then, the popularity of each technology is computed using equation 1 below:

TABLE II. METRICS DESCRIPTION AND DESCRIPTIVE STATISTICS.

Category	Variables	Description	Min	Max	Mean	Median	STDEV
Task	*Award*	Dollars in task description.	0	100000	896.02	500	1744.46
	Type	Type of tasks, i.e. First2Finish, Assembly, Bughunt etc.	1	14	-	-	-
	Duration	Total available time from registration date to submissions deadline	0	73	16.86	9	12.6
	LinkCount	Number of links to detailed requirements or screen mockups in task description	0	58	2.32	2	3.49
Tech	*NumTech*	Count of different technologies for task	1	11	2.73	2	1.20
	MaxTechPop	Most popular technology used within task	0	23.62	17.34	18.41	7.36
	SumTechPop	Total popularity weight of all technologies used for	0	76.98	26.74	23.62	16.50
Network	*PageRank*	The importance of task (node) using PageRank	4.72×10^{-5}	0.11	11.9×10^{-5}	6.05×10^{-5}	157.20
	StrongCompNum	Connectivity of tasks to other nodes (tasks and worker)	0	12688	2518.97	2501.5	1519.61
	Modularity	Strength of network clusters or groups	0	13	4.51	4	3.92
Worker	*NumReg*	Number of registrants that are willing to compete on total number of tasks in specific period	0	3296	12.32	10	48.35
	AvgRely	Average reliability of the involved workers	0	1	0.15	0.12	0.12
	MaxRely	Max reliability of the involved workers	0	1	0.56	0.6	0.28

$$TechPopularity = \sum_{i=1}^{n} \frac{x_i}{K} \quad (1)$$

Where n is the total number of different technologies (i.e. n is 30), and K is the total number of tasks (i.e. K is 4,872). If a task involves a technology i, the corresponding x_i will be 1, otherwise it will be 0. For tasks involving multiple platforms or technologies, we use the sum of the weighted average of *TechPopularity* among all technologies to the task as *sum_Tech_Pop*. Afterward, we get the maximum *TechPopularity* form technologies and add it as additional derived feature *max_Tech_Pop*. We make two basic assumptions in measuring complexity and popularity. One is that the greater the number of platforms/technologies a task requires, the higher its complexity. This is consistent with assumptions and evidence in Mao et al.'s pricing study [6], in which the number of technologies is explicitly modelled as one of the 16 pricing drivers. The other assumption is that the more popular a platform/technology is, the greater pool of potential workers. The evidence is shown in Fig. 3, where the horizontal axis shows the Top-10 technologies in our dataset crowdsourcing tasks, and the vertical axis shows the corresponding frequencies of the number of tasks requiring that technology (in Blue columns) and the number of workers with skills in that technology (in Red columns). There is a clear correlation between these two data series.

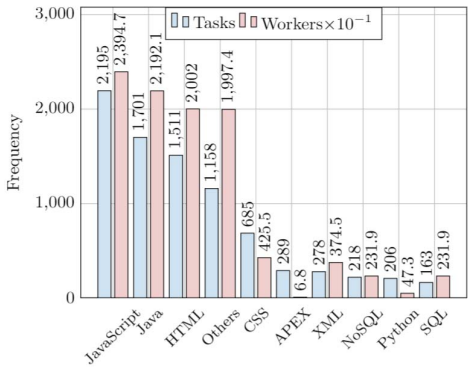

Fig. 3 Popular Technologies Used in TopCoder.

3) Measuring Task Competition Network

To build the graph, we consider unique workers as one type of nodes and tasks as another type of nodes, then define a directed edge from each worker if he/she registered in a task. Because we have two set of nodes, we can be represent as Bipartite graph. The graph contains all the tasks and their registered workers between Jan 2014 and Jan 2015.

Fig. 4 demonstrates an example structure of Biparties graph to measure the network properties of each task. In Fig. 5, the pseudo-code shows the algorithm used for the construction of the graph. Based on network analysis, we will study the influence of network metrics including *Modularity Class*, *PageRank*, and *Strongly Connected Component*.

Modularity characterizes task connection density to help understanding the structure of a large network [19]. PageRank quantifies the importance of nodes in the network [20]. Strongly Connected Components in directed graph adds a new information about different groups of nodes strongly connected based on backtracking the graph [27].

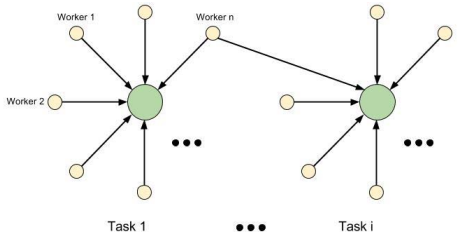

Fig. 4. Illustration of a Bipartite Worker-Task Competition Graph.

```
1:  procedure TasksGraph(V,E)
2:      i ← 1 to 4872
3:      while i ≤ 4872 do
4:          Graph ← taskVertex(i)
5:          n ← count(registredWorkers)
6:          while n ≠ 0 do
7:              if workerVertex(n) ∄ Graph then
8:                  Graph ← workerVertex(n)
9:              end if
10:             Graph ← addEdge(workerVertex(n), taskVertex(i))
11:             n ← n − 1
12:         end while
13:         i ← i + 1
14:     end while
15:     return b
16: end procedure
```

Fig. 5 Pseudo-code for establishing task competition network.

4) Measuring Developer Reliability

For each task, we seek to predict the status from the profiles of the registered pool of workers. Besides the size of the registered pool of workers, i.e. the number of registrants, we also consider the developer reliability factors of the resource pool. Our assumption is that skilled and reliable developers undertaking familiar tasks are more likely to deliver qualified software assets. To measure how reliable developers are, we adapted the definition of developer reliability factor from TopCoder platform. More specifically, "the reliability factor is calculated as the percentage of the last 15 projects that a member registers for in which that same member presents a timely submission that passes review. Based on this definition, we derive two metrics for each task: 1) *AvgRely*, which is the average reliability of all registered workers; and 2) *MaxRely*, which is the highest reliability among all registered workers.

IV. Study Design

To evaluate the proposed failure prediction framework, we design the following research questions (RQs), *in competitive software crowdsourcing tasks:*

RQ1: What are the influencing factors that lead to a task failure? By analyzing different attributes in the dataset, identify the top factors influence the outcome, measure the new defined metrics derived from the raw data. In addition, our study will incorporate the network metrics which are missing in Dubey et al.'s study.

RQ2: Can we build predictive methods for CSD task failure based on the identified factors? We will apply different machine learning techniques using the extracted metrics to

train predictive models and evaluate the prediction performance.

RQ3: To what extent does the purposed predictive model improves the current baselines? We will compare our proposed predictive model with current in-house practice by TopCoder.

A. Dataset

The dataset investigated in our study contains 4,872 competitive software development tasks from Jan 2014 to Jan 2015, extracted from the TopCoder website. Each task is associated with a set of metadata characterizing task attributes such as ID, name, start date, end date, challenge types, platform, technologies involved, etc. The outcome of a task is measured by the number of workers signing up and submitting for the task, evaluated scores of the submission(s), and the end status of the tasks (i.e. completed or cancelled). According to TopCoder's definition, a completed task should have received an accepted submission with a score of at least 75 out of 100. Otherwise, it will be marked with the label of "cancelled" due to either zero submissions or low quality submission score lower than 75. For a completed task, either the Top-1 or Top-2 winners will receive the award according to a detailed arrangement in the task description.

To derive answers to the RQs, we conducted the following data analysis and evaluation experiments.

B. Evaluation Procedures

1) Data Preprocessing

We first filter historical tasks that have incomplete information (e.g. tasks with missing data on important task attributes such as award). Then, in order to better extract knowledge of successful software development tasks, we removed extreme values and outliers such as a task prized $100,000 and attracted 3,296 developers. In fact, this task is divided into 5 "swift debugger" sub-tasks which priced $20,000 divided among multiple winners over a period at 30 days of continuous competition. These tasks, each involving dynamically identifying and awarding multiple winners over a certain period, were removed it to reduce the impact of potential biased or irrelevant empirical knowledge.

2) Data Balancing.

Our dataset was imbalanced with 84.5% of the tasks labelled as successful tasks. To overcome the data imbalance issue, we applied the Synthetic Minority Over-Sampling Technique (SMOTE) [10] to create extra data points based on the minority, which in our case are the failed tasks, by generating synthetic examples. The settings for SMOTE algorithm for our predictive model was as followings. The number of minority class samples is 754; Amount of SMOTE 431%; Number of nearest neighbors was k=5. Based on the dataset the values of SMOTE chosen to make the number of task failure around 50% of the trained dataset.

3) Data Transformation

We adopted some data preprocessing operations to address the quality issues of the data.

Randomize. We applied a random shuffle to the order of instances to eliminate any order issues from the generated failed data points that would affect the machine learner performance.

Normalization. We normalized each attribute to the 0-1 intervals to reduce the effect of different value scales.

4) Learner Selection

We ran the dataset with the identified attributes in Table II and the target feature was the task outcome, in which the value either success or failed, using a set of popular machine learners used in defect prediction research, including Random Forest [28], Naïve Bayes (NB) [29], Logistic Regression, K-Nearest Neighbor (KNN, i.e. K=10 in our study) [25], and StackingC (Ensemble Method) [21]. Based on Hussain et al. conclusion [20], we used Stacking method by setting the base-level classifier with (RF, NB, KNN) and Logistic Regression model as meta classifier.

5) Model Evaluation

Lastly, we applied a 10-fold cross validation to split the dataset into training sets and testing sets. We trained the five learners on the training set, and then applied the trained model to the test set. In this paper, we employ Precision, Recall, and the F-measure that is the harmonic mean of Precision and Recall, to assess the performance of the baseline and the failure prediction models. We adopted these three indicators based on the widely cited studies done by Menzies et al. [29]. According to definitions of these three indicators, we favor prediction results with high Precision, high Recall, and high F-measure. Equations 2, 3 and 4 shows how we calculate these three measures.

$$Precision = \frac{TP}{(TP+FP)} \quad (2)$$

$$Recall = \frac{TP}{(TP+FN)} \quad (3)$$

$$F\text{-}measure = \frac{2 \times Precision \times Recall}{(Precision + Recall)} \quad (4)$$

6) Comparison Baselines

To compare the prediction performance, we used two comparison baselines:

a. TopCoder's in-house predictor

TopCoder currently employs a heuristic-based predictor in which three coded colors (RED, YELLOW, and GREEN) to project task outcomes; it involves three simple rules with respect to the sum of the reliability ratings of all registered workers for a task. More specifically, a worker's reliability rating is a numeric rating between 0 and 1, accounting for the percentage of a worker's successful submissions over his/her last 15 registered tasks. If the reliability ratings sum is greater than or equal to 2, the predictor produces a GREEN label, as an indicator of success for the task to the requester; if it is less than 1, the predictor produces a RED indictor, representing a most likely failed task. While YELLOW is the result in-between of 1 and 2, which is an uncertain situation for a task status. We implement the

TABLE III. DIFFERENT ATTRIBUTE SELECTION AND ATTRIBUTE RANK.

Metric	Corr.	Rank	Gain. Ratio	Rank	Info. Gain	Rank	Symm. Uncer.	Rank	Overall Rank
AvgRely	0.31316	1	0.10542	4	0.55885	1	0.17738	3	1
MaxRely	0.25547	2	0.10403	5	0.52526	2	0.17367	4	2
LinkCount	0.04174	7	0.11529	2	0.38853	4	0.17781	2	3
NumReg	0.08237	5	0.10061	6	0.52156	3	0.16869	6	4
NumTech	0.04351	12	0.1255	1	0.31337	9	0.17922	1	5
Modularity	0.01791	5	0.0989	7	0.33595	8	0.15281	7	6
MaxTechPop	0.01601	13	0.11522	3	0.34809	6	0.17313	5	6
Award	0.17514	3	0.06206	9	0.2468	10	0.09918	10	8
Duration	0.04026	8	0.01	8	0.36274	5	0.15017	8	9
Type	0.11674	4	0.01705	11	0.03093	11	0.02198	11	10
SumTechPop	0.01304	12	0.06766	9	0.33685	7	0.11269	9	10
PageRank	0.01811	9	0.01069	12	0.0231	12	0.01462	12	12
StrongCompNum	0.00926	13	0.0084	13	0.0086	13	0.0085	13	13

TopCoder predictor and use it as one of the comparison baselines. Then we compare its performance with our approach using different learners.

b. Prediction Models in Dubey's Study

Dubey's study provided a top five influence factors with statistical exploration for the dataset from TopCoder between 06/01/2012 to 12/31/2014. The study used Award, Task Category, Duration, Number of Registrants, Number of Technologies, and Maximum Reliability of workers, which are also used in our model. In addition, they also included the winners' placements, the time for the first submission, and the quality of the submissions. We don't employ these in our model because such information would not be available until task completion. The only reported best performance measure in Dubey's Study is an 89% of F-measure from a Random-Forest predictor. We consider it as the second baseline for our study.

V. RESULTS

A. Answer to RQ1: Influencing Factors

To identify the most influential factors of task outcomes, we apply four different feature ranking techniques including correlation analysis (Corr.), Gain Ratio, Information Gain (Info. Gain), and Symmetrical Uncertainty (Symm. Uncer.). The result is summarized in Table III. According to the feature selection results, which indicate that among the investigated factors, the top influential factors are the average of reliability scores for workers (AvgRely); the highest reliable worker score (MaxRely); links in the description (LinkCount); the number of registered workers (NumReg); the number of technologies required (NumTech); and the modularity of competition network (Modularity), and so on. Next, we briefly summarize these factors.

1) Worker Reliability (AvgRely and MaxRely)

Table III suggests that crowdsourcing success is largely associated with worker reliability, with AvgRely and MaxRely rank as the 1st and 2nd influencing factors. The higher the average reliability of registered workers is, the greater probability that the task is successfully completed. While MaxRely refers to the highest reliability among all registered workers, it is less significant than the average reliability in differentiating cancelled tasks from successful ones.

2) LinkCount

LinkCount is ranked as the third factor for the task success. Typically, additional description or specification details are provided as links embedded in task description, to facilitate worker understanding.

3) Technologies (NumTech and MaxTechPop)

The results also show that two of the technological factors, *i.e.* NumTech and MaxTechPop, are ranked as 5th and 7th in the list. This confirms our assumption that the more popular the technology is, the greater possibility to succeed due to potentially larger worker supply pool. However, the positive correlation between NumTech and task success is counter-intuitive. A possible explanation would be that the multiple technology complexity has been taken into consideration into motivating factors such as higher award in attracting more capable workers. On the other hand, it is not necessary to derive that a low popular technology will be subject to greater failure chance. In one case, we found 78% of tasks (i.e. how many tasks in total 287) require APEX technology had succeed despite the low supply of workers (68 workers) that registered in similar tasks that require APEX technology.

4) Network

Modularity is ranked as the 6th factor, while the other two network factors, i.e. PageRank and StronglyConnected-Components, are ranked the last two in the list.

5) Task (Award, Duration, Type)

Award is ranked 8th in our study, while it is ranked as the 4th factor in Dubey's study. Duration is ranked 9th in our study, while it is ranked as the 5th in Dubey's study. Lastly, Dubey's model ranks the challengeType as the 2nd factor; while it is only ranked 10th in our study. The results indicates that some additional factors proposed in our study show a greater influence in task success.

B. Answer to RQ2:

The results of our empirical study are presented in Table V. In general, an encouraging performance results in the task failure predictive models. With each machine learner we include the precision, recall and f-measure for each label (i.e. failure vs. completion). The top-3 failure learners for each measure are marked in bold in Table V.

TABLE V. SUMMARY OF PREDICTIVE PERFORMANCE

Method	Label	Precision	Recall	F-Measure
Naïve Bayes	Failure	**0.999**	0.812	**0.896**
	Completion	0.841	***0.999***	***0.913***
	Average	0.920	0.905	0.905
Random Forest	Failure	0.988	**0.819**	**0.896**
	Completion	***0.845***	0.990	***0.912***
	Average	0.917	0.905	0.904
SVM	Failure	**1.000**	0.801	0.889
	Completion	0.833	***1.000***	0.909
	Average	0.917	0.900	0.899
Logistic	Failure	0.965	**0.831**	0.893
	Completion	***0.851***	0.970	0.907
	Average	0.908	0.900	0.900
KNN (K=10)	Failure	0.993	0.791	0.881
	Completion	0.826	0.995	0.903
	Average	0.910	0.893	0.892
KNN (K=5)	Failure	0.981	0.812	0.889
	Completion	0.840	0.984	0.906
	Average	0.910	0.898	0.898
KNN (K=3)	Failure	0.956	0.828	0.887
	Completion	***0.848***	0.962	0.901
	Average	0.902	0.895	0.894
StackingC	Failure	**0.997**	0.814	**0.896**
	Completion	0.842	***0.998***	***0.913***
	Average	0.920	0.906	0.905
Baselines				
TopCoder Predicator	Completion	*0.660*	*0.620*	*0.639*
Dubey's model (RF)	Completion	-	-	*0.89*

According to precision, which represents the proportion of the predict outcome over the actual outcome, all learners can predict "failure" class with a precision greater than 95.6%, with the highest precision of 99.9% from Naïve Bayes. This means that more than 95.6% of all predicted failures are actually failures. According to recall, the accuracy is also very good (i.e. ranging from 79.1% from KNN with k=10, to 83.1% from Logistic). Though this is also considered high performance, it indicates that there are about 20% of actual failures are missed by the prediction models. According to F-measure, the harmonic mean of precision and recall, we observe that different learner performs between 88.1% from KNN(k=10) and 89.6% from NB, RF, and StackingC. It's worth noting that StackingC, as an ensemble approach, does not outperform individual classifiers. This is different from the conclusion reported in [20]. Nonetheless, Naïve Bayes, StackingC, and Random Forest are among the Top-3 learners in our case, outperforming the other learners.

Even though, our scope is to purpose a failure prediction model. We have to introduce the completion-prone in our study, Table V, to accurately compare our learner's performance with to TopCoder Predictor the first baseline, and Dubey's best model, Random Forest, the second baseline. This will be detailed in the next section.

C. *Answer to RQ3:*

Note that the performance for predicting "completion" class is also reported in Table V in order to compare with the selected baselines. The reason is that only performance for completion class prediction is reported in Dubey et al.'s study [12]. The top-3 completion learners for each measure are marked in italic bold, red colored, and gray shaded.

As shown in the bottom part of Table V, all learners show improved performance, in terms of F-measure, over the TopCoder Predictor. For the first baseline of TopCoder predictor, the improvement is 19.1% (85.1% vs. 66%) in precision, 38% (100% vs. 62%) in recall, and 27.4% (91.3% vs. 63.9%) in F-Measure. For the second baseline, the best learner from Dubey et al.'s study, an improvement of 2.3% in F-measure is concluded. Since no other measures is reported in Dubey's study, we cannot compare precision and recall measures.

VI. DISCUSSION

A. *Predictability of Software Crowdsourcing Tasks*

This study provides empirical evidence on software crowdsourcing development task failure predication. Using the proposed failure prediction framework, esp. extracting features characterizing various factors from TASK, TECH, WORKER and NETWORK categories, the achieved accuracy was improved noticeably comparing to the TopCoder prediction baseline and slight improvement to Dubey's results. The additional factors identified in our study, such as task competition network and technology popularity metrics help to contribute to such improvement.

As further investigation, we also analyzed the predictability of task completion scores using the dataset and the five learners. The results in Table VI shows that the best predictor Naïve Bayes can predict the final score with a Mean Absolute Error (MAE) of 0.095, which is considered very accurate and encouraging. Based on learned proposed in this study, task requesters can have access to more insights on the likelihood of potential task outcomes based on ongoing dynamics of worker participations. As automatic monitoring and control support aid, if any failure proneness detected, task requesters can be warned and mitigation actions can be discussed and taken place accordingly.

TABLE VI. COMPARISON OF MAE IN FINAL SCORE PREDICTION.

Method	MAE
Naïve Bayes	*0.095*
SVM	*0.100*
StackingC	0.157
KNN (K=3)	0.155
Random Forest	0.165
KNN (K=5)	0.165
Logistic Regression	0.165
KNN (K=10)	0.184
TopCoder Predictive	*0.379*

Considering the largely reported promising benefits of CSD such as 30% - 80% cost reduction compared to in-

house development or outsourcing, or the purposed 5 to 8 times lower defect rate as compared with traditional software development practices [1], our results provide a complementary perspective and method to support better planning and managing crowdsourced software development.

B. Practical Insights

Our results indicate that crowdsourcing failure can be predictable. This is encouraging evidence in that there is a value in developing supporting tools for monitoring and controlling the risks of crowdsourced software development projects. Such supporting tool address the failure risk at the task design phase or during the task registration phase. It is very encouraging that by employing a simpler model, such as the Naïve Bayes learner, we can project failure-prone tasks to 89% f-measure with low MAE of 9.5%.

More practical guidance can be concluded based on the most influential factors of task failure, which includes workers reliability, task size, award, and network degree, are the most influencing factors for crowdsourcing.

1) Recommendation for Task Design

In crowdsourced software development, importance steps in the task design phase consist of task decomposition and micro-task preparation. The empirical results lead us to derive a few insights to guide decisions in the task design phase, which we list as follows:

- It is observed that workers with higher reliability scores lead to task success. For that, attracting a reliable worker to the task is an influential factor for successful outcome.
- Task award is a not a major factor to attract workers, as it is only ranked as #8 factors, as shown in Table II.
- Decomposing a task into a shorter set of tasks to reduce the risk of cancellation. The average duration of successful tasks is 14 days, while that of cancelled tasks is 18 days. More than half of the successful tasks are scheduled for 1 day.
- Incorporate analytical-based learners can help task requestors monitor and predict failure risk and improving task design.

2) Recommendation for Managing a Well-structured Worker Community

Understanding the developer's interaction, especially, on how they register in different tasks and their effects on each other that in which is captured by network analysis. For that, ensure a highly reliable worker who participates in task gives a greater chance of success for a crowdsourced software task. Thus, identifying and mobilizing the best and most reliable worker resource is essential. We offer insights from both a pre-registration resource identification perspective and a post-registration risk monitoring perspective.

- *Pre-registration*: In a recent study, Mao et. al. proposed a recommendation system [18] that can learn from task similarity and competition history and recommended diversity of competition and both reliable workers and participatory workers to guarantee the quality of delivered assets. The results from our study confirms that it is more important to cultivate a diversified yet reliable crowd (i.e. with high AvgRely), rather than the best individual (i.e. MaxRely) in order to ensure greater confidence in crowdsourcing success.
- *Post-registration*: The developer rating and reliability metric system on TopCoder contains much invaluable data to help understand workers participation and abandonment in his/her competition history. Though our proposed task failure prediction method is built based on historical project data, it can be extended and applied in real time cases in order to provide early risk indicators for task requesters.

C. Limitations

There are several limitations to this study. First, the study only focuses on competitive software crowdsourcing tasks on TopCoder platform. There are many non-competitive or collaborative crowdsourcing tasks, which may demonstrate different task organization, pricing patterns, and worker behaviors, etc. Second, there are some competition factors such as number of available tasks to select, number of tasks a worker has, etc. that might also impact on task failure likelihood. These are not included in current study yet and will be one of our future research topics. Third, proposed metrics such as studying of supply and demand for technologies and worker, social network analysis and others need a dedicated study to get more understanding of the factors for task failure. Fourth, regarding internal threats, different settings (e.g., learner, sample size) may lead to different model outputs.

D. Threat to Validity

Two types of threats of validity will be discussed. We used Python Sklearn to perform our empirical study. The results of Random Forest machine learner impacted by imbalanced data in which our study includes 15.4% failure rate. For that, we used SMOTE, an oversampling approach, to balance the dataset. SMOTE generates the new data points based on the historical data points, which in real-world may differ than the ones created using SMOTE.

Even though Naïve Bayes performs the best in our study, the algorithm assumes that all the features independent to predict task outcome [23]. In fact, some features in the study have dependencies such as the average of reliability and the maximum of reliability.

Many other possible additional factors could affect the task completion status, such as how familiar or experienced individual workers are to the task? To what degree individual sign-up worker can handle multi-tasking? How many other similar tasks are simultaneously competing resources with the task? How about other semantic factors

from task description? The effects of such factors are not considered in current framework.

Threats to external validity by considering the generalization of the study to other types of crowdsourcing tasks, or other platforms than TopCoder. Our research is focused on software development crowdsourcing tasks and in particular TopCoder. Other platforms offer a different flow of requesting the task and evaluating the workers. Thus, the application of the proposed framework might need to be carefully examined to accommodate such differences.

VII. Related Work

A. CSD Success Factors

Messinger highlighted three elements of successful software crowdsourcing quality community, right incentives, and trust and transparency [17]. Software crowdsourcing platforms attracts developers with certain skillsets to participate in the open-call tasks, and the quality of workers' leads to success in the crowdsourced tasks [7, 8]. CSD platforms, such as TopCoder, typically measure the quality of each worker by assigning a community rating score based on his past behaviors, from both quality and quantity aspects. Additionally, a Reliability score is employed by TopCoder to indicate the reliability and consistency of a worker's performance in his/her most recent 15 tasks. These types of scoring schemes provide a good evaluation for the quality workers in the platforms.

Identifying the motivation for workers to participate and submit for tasks is one of the key concept to attract them in proposed task. The incentives for the workers are not limited to monetary reward. Borst [19] stated that the extrinsic motivations for workers effected the participation, behavior and performance positively by the existence of the reward. An evidence, in some crowdsourced tasks in TopCoder with zero monetary reward shows a success outcome. For that, highly self-motivated workers submit for such tasks based on Borst study [19]. For increasing amount of award, Yang [5] shows that number of registered workers in similar tasks shapes an inverted U-curve. For that, a careful chose of right incentives is not exclusive to the task reward or requirement; it's also depends on the workers' motivations to participate on certain task.

The uncertainty of task outcome raises the trust issues between workers and task owner. From the task requester's perspectives, it is challenging to identify best workers for their tasks, and even more challenging to monitor risks related to workers reliability shortfalls. CSD platforms employ peer-review processes to evaluate workers' submissions, there might be possibilities that the review processes can be strategically manipulated by workers or task owners [24]. In addition, workers provided information about their teammates or opponents outperformed the anonymous condition in task delivery [4].

B. CSD Decision Support

To address the potential task completion risk, Dubey et al. identify the top five influence factors. By using a different machine learning algorithms with exploratory analysis. The study highlighted these factors considering information gain weight for ranking the factors. The results show the most influence factors for task completion are rating of register workers, type of the task, winner's prize and duration of the task respectively. Despite that, the importance on analyzing task features such as, worker reliability, incentives, technologies, complexity and duration helps the requester to propose a well-structured task. On one hand, task features such as task complexity and task incentives consider to be motivations for workers to participate in task. On the other hand, attracting strong developers with high rating and reliability score effects on the success of a task. Thus, capturing all these features can be achieved by representing the workers and tasks in graph and extract the properties to predict the task status accurately, in which not been discussed in the study. Actually, the best performance algorithm from Dubey's study will be consider as baseline for our proposed model.

Another type of CSD decision support is providing a relevant pool of workers for purposed new tasks [18] [9] or recommending appropriate tasks for workers [7] [16]. Existing studies deriving workers quality based on their reputation and expertise based on their earlier tasks participation [7] [13] [15] [18]. As an efficient way to identify qualified workers, Mao et al. proposed a content-based recommendation system to automatically match tasks and developers based on historical data [18]. The system learns from the winning history for recommending reliable developers and learns from registration history to suggest suitable participants for available tasks. Yang et al. proposed a dynamic decision support to help worker choosing the suitable tasks [7]. For matching the best workers for available tasks, Mao et al. introduce a decision support of Prestige Network Enhanced Developer-Task Matching for Crowdsourced Software Development [9].

C. Social Network Analysis in CSD

Social network analysis techniques quantify the social behaviors represented in a set of nodes and edges [19] [11]. Existing studies have applied social network analysis in CSD in the competition network between developers [14]. PREM examines the developers' historical task data and rating to construct a competition network between winners and registered workers aiming to match suitable workers with tasks [9]. In like manner, Zhang et al. build a competition network and concluded a positive correlation between the social network properties of developers and the outcome of the projects [14].

In our study, we will construct the crowdsourcing participatory network to capture the tasks and developers in

which we include the social network metrics for each task as additional features to build failure prediction models, to improve the performance of task failure prediction.

VIII. CONCLUSIONS

In the context of crowdsourced software development (CSD), projects are typically organized into dozens or hundreds of micro-tasks, it is not feasible to manually monitor the ongoing progress status of large number of crowdsourced tasks. Despite the increasingly reported benefits associated with CSD, one of the major practical concerns is the limited visibility and control over task progress. This study seeks to precisely detect cancellation-prone tasks and thus helps to take management actions to prevent potential delay. We developed empirically based metrics and employed machine learning techniques to automatically predict task failure risk based on historical data. The Study's main results include: 1) Workers reliability, links in the description, number of registered workers, number of required technologies, and task-workers network modularity are the most influencing factors for predicting crowdsourcing failure; 2) The top-three learners for task failure are Naïve Bayes, Random Forest, and StackingC, with precision above 98.8%, recall above 81.2%, and F-measure above 91.2%; and 3) The proposed best learners significantly outperform the two baseline models in our evaluation. Future research direction includes broader data collection and evaluation, as well as utilizing text mining approaches to understanding the effect of task description on task failure prediction.

REFERENCES

[1] Lakhani, Karim R. "Managing communities and contests to innovate with crowds." *Revolutionizing Innovation: Users, Communities, and Open Innovation* (2016): 109.

[2] TopCoder – A Platform Overview. Retrieved from NASA.gov *http://www.nasa.gov/pdf/651447main_TopCoder_Mike_D1_830am.pdf*

[3] Prikladnicki, Rafael, Leticia Machado, Erran Carmel, and Cleidson RB de Souza. "Brazil software crowdsourcing: a first step in a multi-year study." In *Proc. of the 1st Int. Workshop on CrowdSourcing in Softw. Eng.*, pp. 1-4. ACM, 2014.

[4] Bernstein, Michael S *et al*. "Soylent: a word processor with a crowd inside." *Commun. of the ACM* 58, no. 8 (2015): 85-94.

[5] Yang, Ye, and Razieh Saremi. "Award vs. Worker Behaviors in Competitive Crowdsourcing Tasks." In *Empirical Softw. Eng. and Measurement (ESEM), 2015 ACM/IEEE Int. Symp. on*, pp. 1-10. IEEE, 2015.

[6] Mao, Ke, Ye Yang, Mingshu Li, and Mark Harman. "Pricing crowdsourcing-based software development tasks." In *Proceedings of the 2013 international conference on Software engineering*, pp. 1205-1208. IEEE Press, 2013.

[7] Yang, Ye, Muhammad Rezaul Karim, Razieh Saremi, and Guenther Ruhe. "Who Should Take This Task?: Dynamic Decision Support for Crowd Workers." In *Proc. of the 10th ACM/IEEE Int. Symp. on Empirical Softw. Eng. and Measurement*, p. 8. ACM, 2016.

[8] Kittur, Aniket, Ed H. Chi, and Bongwon Suh. "Crowdsourcing user studies with Mechanical Turk." In *Proc. of the SIGCHI conf. on human factors in computing syst.*, pp. 453-456. ACM, 2008.

[9] Mao, Ke, *et al*. "PREM: Prestige Network Enhanced Developer-Task Matching for Crowdsourced Software Development." (2016).

[10] Chawla, Nitesh V., Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique." *J. of artificial intel. research* 16 (2002): 321-357.

[11] Leenders, Roger TAJ, and Wilfred A. Dolfsma. "Social networks for innovation and new product development." *J. of Product Innovation Manage.* 33, no. 2 (2016): 123-131.

[12] Dubey, Alpana, *et al*. "Dynamics of software development crowdsourcing." In *Global Softw. Eng. (ICGSE), 2016 IEEE 11th Int. Conf. on*, pp. 49-58. IEEE, 2016.

[13] Tan, Wei, M. Brian Blake, Iman Saleh, and Schahram Dustdar. "Social-network-sourced big data analytics." *IEEE Internet Computing* 17, no. 5 (2013): 62-69.

[14] Zhang, Hui, Yuchuan Wu, and Wenjun Wu. "Analyzing developer behavior and community structure in software crowdsourcing." In *Information science and applications*, pp. 981-988. Springer, Berlin, Heidelberg, 2015.

[15] Karim, Muhammad Rezaul, David Messinger, Ye Yang, and Guenther Ruhe. "Decision Support for Increasing the Efficiency of Crowdsourced Software Development." *arXiv preprint arXiv:1610.04142* (2016).

[16] Tomek, Ivan. "An experiment with the edited nearest-neighbor rule." *IEEE Transactions on systems, Man, and Cybernetics* 6 (1976): 448-452.

[17] Messinger, D. (2016). *Elements of Good Crowdsourcing*. Keynote presented at 3rd International Workshop in Austin, Texas.

[18] Mao, Ke, Ye Yang, Qing Wang, Yue Jia, and Mark Harman. "Developer recommendation for crowdsourced software development tasks." In *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*, pp. 347-356. IEEE, 2015.

[19] Borst, Irma. *Understanding Crowdsourcing: Effects of motivation and rewards on participation and performance in voluntary online activities*. No. EPS-2010-221-LIS. 2010.

[20] Hussain, Shahid, *et al*. "Performance evaluation of ensemble methods for software fault prediction: An experiment." In *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*, pp. 91-95. ACM, 2015.

[21] Seewald, Alexander K. "How to make stacking better and faster while also taking care of an unknown weakness." In *Proceedings of the nineteenth international conference on machine learning*, pp. 554-561. Morgan Kaufmann Publishers Inc., 2002.

[22] Archak, Nikolay. "Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on TopCoder. com." In *Proceedings of the 19th international conference on World wide web*, pp. 21-30. ACM, 2010.

[23] Khoshgoftaar, Taghi M., Moiz Golawala, and Jason Van Hulse. "An empirical study of learning from imbalanced data using random forest." In *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, vol. 2, pp. 310-317. IEEE, 2007.

[24] Kamar, Ece, and Eric Horvitz. "Incentives for truthful reporting in crowdsourcing." In *Proc. of the 11th int. conference on autonomous agents and multiagent syst.-volume 3*, pp. 1329-1330. Int. Foundation for Autonomous Agents and Multiagent Systems, 2012.

[25] Blondel, Vincent D., *et al*. "Fast unfolding of communities in large networks." *J. of statistical mechanics: theory and experiment*2008, no. 10 (2008): P10008.

[26] Page, Lawrence, *et al*. *The PageRank citation ranking: Bringing order to the web*. Stanford InfoLab, 1999.

[27] Tarjan, Robert. "Depth-first search and linear graph algorithms." *SIAM journal on computing* 1, no. 2 (1972): 146-160.

[28] Lessmann, Stefan, Bart *et al*. "Benchmarking classification models for software defect prediction: A proposed framework and novel findings." *IEEE Trans. on Softw. Engi.* 34, no. 4 (2008): 485-496.

[29] Menzies, Tim, Jeremy Greenwald, and Art Frank. "Data mining static code attributes to learn defect predictors." *IEEE transactions on software engineering* 33, no. 1 (2007): 2-13.

[30] Begel, Andrew, Jan Bosch, and Margaret-Anne Storey. "Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder." *IEEE Software* 30, no. 1 (2013): 52-66.