Impact of Task Cycle Pattern on Project Success in Software Crowdsourcing

Razieh Saremi[1], Marzieh Lotfalian Saremi[2], Sanam Jena[1], Robert Anzalone[1], and Ahmed Bahabry[1]

[1] Stevens Institute of Technology, Hoboken NJ, USA
[2] Concordia University, Montreal, Qc, Canada
{rsaremi,sjena,ranzalon,abahabry}@stevens.edu, m.lotfa@encs.concordia.ca

Abstract. Crowdsourcing is becoming an accepted method of software development for different phases in the production lifecycle. Ideally, mass parallel production through Crowdsourcing could be an option for rapid acquisition in software engineering by leveraging infinite worker resource on the internet. It is important to understand the patterns and strategies of decomposing and uploading parallel tasks to maintain a stable worker supply as well as a satisfactory task completion rate.
This research report is an empirical analysis of the available tasks' lifecycle patterns in crowdsourcing. Following the waterfall model in Crowdsourced Software Development (CSD), this research identified four patterns for the sequence of task arrival per project: 1) Prior Cycle, 2) Current Cycle, 3) Orbit Cycle, and 4) Fresh Cycle.

Keywords: Crowdsourcing · Topcoder · Task Lifecycle · Task Failure Ratio.

1 Introduction

The interest in crowdsourcing software development(CSD) is rapidly increasing in both industry and academia. In crowdsourcing, jobs that were traditionally done in-house would be distributed among a large, distributed group of crowd workers[7]. Software projects are using crowdsourcing methods in different phases of software design and production[28]. While understanding the

patterns and strategies of decomposing and scheduling crowdsourced are important to maintain stable worker supply as well as satisfactory task completion rate, much of existing studies are only focusing on individual task level, such as task pricing, task similarity, and task diversity [5][1][16][22], and worker recommendation and behavior models[12][25][24].

Crowd workers usually choose to register, work, and submit for certain tasks with the satisfactory monetary prize and comfortable level of dedicated effort, as monetary prize typically represents a degree of task complexity as well as required competition levels [5][2]. Although sometimes Award is simply representing a specifically required skill to perform the task, it is one of the main factors influence crowd software workers in terms of the number of registrants

and consequently, the number of submissions [32]. Therefore, pricing tasks could be a huge challenge in decomposing the projects to mini-tasks and time of uploading them in the platform, yet by uploading more number parallel tasks, crowd workers would have more choice of utilized tasks to register for and consequently the chance of receiving more number of completed tasks is higher[26]. Since existing studies on general crowdsourcing reported limited or unpredictable results [5][2][12], it is a good opportunity to focus on parallelism on uploading the same project tasks.

For software managers, utilizing external unknown, uncontrollable, crowd workers would put their projects under greater uncertainty and risk compared with in-house development [32][12]. Understanding crowd worker's sensitivity to the project stability and failure rate becomes extremely important for managers to make trade-offs among cost-saving, degree of competition, and expected quality in the deliverable[32].

To address these challenges, existing methods have explored various methods and techniques to bridge the information gap between demand and supply in crowdsourcing-based software development. These includes: studies towards developing better understanding of worker motivation and preferences in CSD [5][6][4][32], studies focusing on predicting task failure [9][10]; studies employing modeling and simulation techniques to optimize CSD task execution processes[21][23][29]; and studies for recommending the most suitable workers for tasks [31] and developing methods to create crowdsourced teams[30] [33]. However, there is a lack of study on the task execution flow in the field of software crowdsourcing.

To develop a better understanding of crowdsourcing-based software projects, this research reports an empirical study on analyzing Topcoder [1], largest software development crowdsourcing platform with an online community of 750000 Crowd Software workers. Topcoder intensively leverages crowdsourcing throughout the product implementation, testing, and assembly phases.

Topcoder started to explore crowdsourcing tasks in the form of competitions for software development, in which workers would independently create a solution and the winner will be chosen[25].

The remainder of this paper is structured as follows. Section II presents the background and review of available works. Section III outlines our research design. Section IV presents the empirical result. Section V discusses the findings and limitations and Section VI presents the conclusion and outlines several directions for future work.

2 Research Background

2.1 Crowdsourced Platform Workflow

A successful crowdsourcing platform contains three determinants: the characteristics of the project; the composition of the crowd; and the relationship among

[1] https://www.topcoder.com/

key players [13]. A systematic development process in a crowdsourcing platform starts from a requirements phase, where the project goals, task plan, and budget estimation are recognized. This will be performed through communication between the project manager, who may come from the crowd or the platform, and the requester, who pays for the solutions offered by the crowd. The outcome will be a set of requirements and specifications. These requirements are used as the input for the future architecture phase, while the application is decomposed into multiple components [17]. In CSD workflow , the task owner divides the project into many small tasks, prepares task descriptions, and distributes tasks through the platform. Each task is tagged with a pre-specified prize as the award [14][32] to winners and a required schedule deadline to complete. On average, most of the tasks have a life span of 2-4 weeks from the first day of registration.

Crowd software workers browse and register to work on selected tasks, and then submit the work products once completed. After workers submit the final submissions, the files will be evaluated by experts and experienced workers, through a peer review process, to check the code quality and/or document quality [13]. The number of submissions and the associated evaluated scores replicate the level of success in task success. In TopCoder, usually the award goes to the top 1 or 2 winners with the highest scores. If there are zero submissions, or no qualified submissions, the task will be treated as starved or cancelled. Figure 1 illustrates the CSD flow.

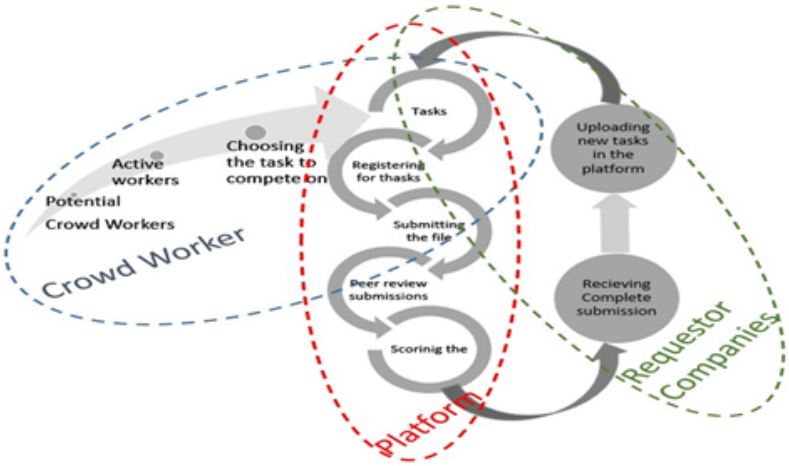

Fig. 1. Crowdsourcing Software Development Flows[32]

2.2 Task Decomposition in Crowdsourcing

In general, projects in crowdsourcing can be decomposed and executed in both independent and dependent tasks. There are two methods of task decomposi-

tion: 1) horizontal task decomposition for independent subtasks and 2) vertical task decomposition for dependent sub-tasks[8]. In the horizontal decomposition method, workers dedicate efforts independently to their own subtasks for individual utility maximization, while in the vertical decomposition method, each subtask takes the output from the previous subtasks as input, and therefore the quality of each task is not only related to the worker's effort but also associated with the quality of the previous tasks[8].

Crowdsourcing complex tasks, i.e. software development tasks, generate heavy work-loads and require dedicated resources with high skill sets, which limit the pool of potential workers. In order to expand the qualified labor pool, it is essential to decompose software engineering tasks into smaller pieces. However, software engineering tasks are often concerned with their specific contexts, for which decomposition may be complicated. This fact opens a discussion on different ways of decomposition based on a hierarchy of workflow [27].

The key factors of decompositions considered in this research are: smaller size of micro-tasks, larger parallelism, reducing time to market, and a higher probability of communication overhead. The most common method in decomposing microtasks is asking individual workers to work on a task of specific artifact. This method will lead to some natural boundaries for software workers and there may be a need of defining new boundaries as well.

2.3 Task Flow in Crowdsourcing

Different characteristics of machine and human behavior create delays in product release[20]. This phenomenon leads to a lack of systematic processes to balance the delivery of features with the available resources [20]. Therefore, improper scheduling would result in task starvation [5]. Parallelism in scheduling is a great method to create the chance of utilizing a greater pool of workers [18, 26]. Parallelism encourages workers to specialize and complete tasks in a shorter period. The method also promotes solutions that benefit the requester and can help researchers to clearly understand how workers decide to compete on a task and analyze the crowd workers performance [5]. Shorter schedule planning can be one of the most notable advantages of using CSD for managers [11].

Batching tasks in similar groups is another effective method to reduce the complexity of tasks and it can dramatically reduce costs[15]. Batching crowdsourcing tasks would lead to a faster result than approaches which keep workers separate[3]. There is a theoretical minimum batch size for every project according to the principles of product development flow [19]. To some extent, the success of software crowdsourcing is associated with reduced batch size in small tasks.

3 Empirical Study Design

3.1 Empirical Analysis

To develop better understanding on the dynamic patterns in task supply and execution, we formulated research questions below and investigated the different task cycle patterns in CSD and relationships among them in quantitative

manners. The required steps to answer the the proposed research questions is illustrated in Fig 2.

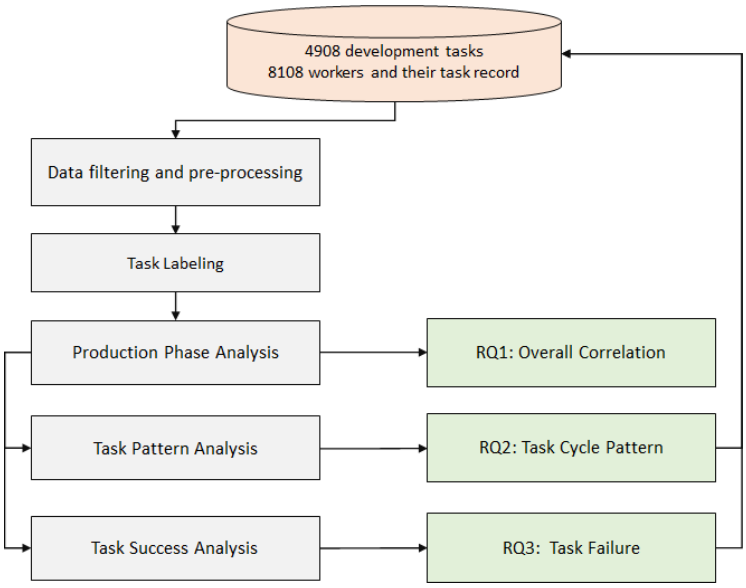

Fig. 2. Main flow of the proposed framework and relationship to research questions

To investigate the evaluation framework, the following research questions were formulated and studied in this paper:

1. *RQ1 (Overall Correlation)*: How does different task production phases correlate with task success?
   This research question aims at providing general overview of task success per software production phases in CSD platform;
2. *RQ2 (Task Cycle Pattern)*: Is there any task cycle pattern in CSD?
   Understanding task cycle patterns in CSD can be good measure to indicate project success;
3. *RQ3 (Task Failure)*: How does different task cycle patterns impact task Failure?
   The ratio of receiving not valid submission per task phase per task cycle pattern represents task success per identified task cycle.

3.2 Dataset

The dataset from TopCoder contains 403 individual projects including 4,907 component development tasks (ended up with 4,770 after removing tasks with incomplete information) and 8,108 workers from January 2014 to February 2015

(14 months). Tasks are uploaded as competitions in the platform, where Crowd software workers would register and complete the challenges. On average, most of the tasks have a life cycle of one and half months from first day of registration to the submission's deadline. When the workers submit the final files, it will be reviewed by experts to check the results and grant the scores.

Table 1. Summary of Metrics Definition

Metrics	Definition
Task registration start date (TR)	The first day of task arrival in the platform and when workers can start registering for it. (mm/dd/yyyy)
Task submission end date (TS)	Deadline by which all workers who registered for task have to submit their final results. (mm/dd/yyyy)
Task registration end date (TRE)	The last day that a task is available to be registered for. (mm/dd/yyyy)
Monetary Prize (MP)	Monetary prize (USD) offered for completing the task and is found in task description. Range: (0, ∞).
Technology (Tech)	Required programming language to perform the task. Range: (0, # Tech)
Platforms (PLT)	Number of platforms used in task. Range: (0, ∞).
Task Status	Completed or failed tasks
# Registration (R)	Number of registrants that sign up to compete in completing a task before registration deadline. Range: (0, ∞).
# Submissions (S)	Number of submissions that a task receives before submission deadline. Range: (0, # registrants].
# Valid Submissions (VS)	Number of submissions that a task receives by its submission deadline that have passed the peer review. Range: (0, # registrants].

The dataset contains tasks attributes such as technology, platform, task description, monetary prize, days to submit, registration date, submission date, and the time-period (month) on which the task was launched in the platform.Then, we used expert based method and labeled associated phase with each tasks. The tasks attributes used in the analysis are presented in top section of Table 1.

Moreover, Topcoder clustered different task type to 7 groups as bellow:

1. *First2Finish*: The first person to submit passing entry wins
2. *Assembly Competition*: Assemble previous tasks
3. *Bug Hunt*: Find and fix available Bugs
4. *Code*: Programming specific task
5. *UI Prototype*: User Interface prototyping is an analysis technique in which users are actively involved in the mocking-up of the UI for a system.
6. *Architecture*: This contest asks competitors to define the technical approach to implement the requirements. The output is a technical architecture document and finalized a plan for assembly contests.

7. *Test Suit*: Competitors produce automated test cases to validate the quality, accuracy, and performance of applications. The output is a suite of automated test cases.

4 Empirical Results

Overall Correlation (RQ1) It is reported that CSD platforms are following waterfall development model [27]. Therefore, all tasks in the platform will follow development phases of Requirements, Design, Implementation, Testing, and Maintenance one after the other. Figure 3 presents the distribution of failure ratio for different task types per task phase in the task life cycle.

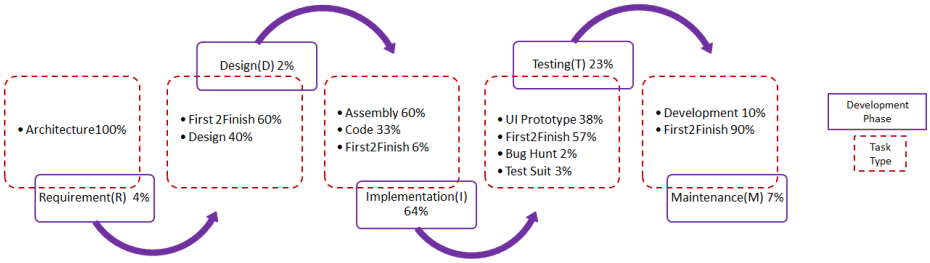

Fig. 3. Distribution of Task Failure Ratio among Different Development Phase in Task Lifecycle

As it is shown in figure 3, highest task failure takes place in the implementation and testing phase with 64% and 23% respectively, while design and requirement are sharing less than 5% of task failure each. One reason can be a lower number of the available task in these phases. Interestingly maintenance is holding 7% of task failure in the platform. Moreover, First2Finish cluster contains tasks from all different development phase. This task type can be assigned to all different phases. As it is shown in figure3, 90% of task failure in maintenance and 57% of task failure in testing phase belong to this task type. While in implementation, only 6% of task failure is under First2Finish task type, and interestingly 60% of task failure happens in assembly tasks.

Task Cycle Pattern (RQ2) Waterfall development model in CSD makes each batch of tasks be always from a prior batch and a fresh batch. Following this sequence of task arrival we identified four cluster of task cycle patterns per project in CSD. In this study, the patterns are named as four clusters of Prior Cycle, Current Cycle, Orbit Cycle and fresh Cycle, which are defined as below:

Figure 4 presents the summary of task cycles in a CSD platform.

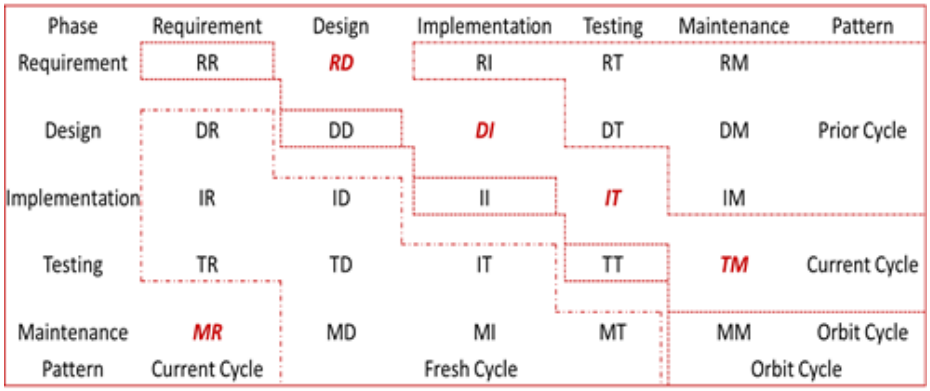

Fig. 4. Summary of Task Cycle in CSD Projects

1. *Current Cycle* is the batch of tasks that are scheduled to complete following the initial task life cycle.
2. *Prior Cycle* is the batch of tasks that went into task life cycle before the batch of current cycle arrives at the platform.
3. *Fresh cycle* is the batch of tasks that will start their life cycle after the current cycle arrives at the platform.
4. *Orbit Cycle* is the batch of tasks that all following the same development phase in the task life cycle.

Task Failure (RQ3) Our analysis shows that on average 44% of tasks in Topcoder are in Fresh Cycle while only 4% of tasks are in the Prior cycle. Also, only 15% of tasks are in Current Cycle and 37% of tasks are in Orbit Cycle. 18% and 20% of task failure is respectively associated with Orbit Cycle and Fresh Cycle. Figure 5 illustrates the details of task failure pattern in different task cycles.

5 Discussion

5.1 Empirical Findings

As it is shown in figure 3, highest failure ratio happened in the same development phase. This can be the impact of task similarity. Our empirical result support that failure mostly happened due to receiving no submission. These observations raises the importance of studying both workers behavior and task similarity in details.

Moreover, identifying available task cycle in CSD revealed that the highest level of success is the result of task prior cycle while least level of task success is the result of fresh cycle task.

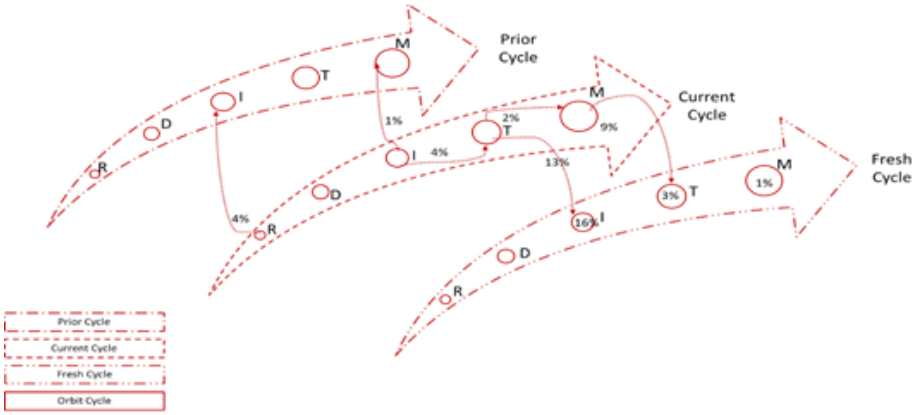

Fig. 5. Task Failure Pattern in Different Task Cycle

5.2 Threats to Validity

First, the study only focuses on competitive CSD tasks on the TopCoder platform. Many more platforms do exist, and even though the results achieved are based on a comprehensive set of about 5,000 development tasks, the results cannot be claimed to be externally valid. There is no guarantee the same results would remain exactly the same in other CSD platforms.

Second, there are many different factors that may influence task success, and task completion. Our task failure probability-focused approach is based on known task attributes in TopCoder. Different task failure probability-focused approaches may lead us to different, but similar results.

Third, the result is based on tasks description and completion only. Project level description and limitations are not considered in this research. In the future, we need to add this level of research to the existing one.

6 Conclusion and future work

To understand the probability of a tasks success in a crowdsource platform, one should understand the task cycle patterns in the platform. This research investigated available task cycle patterns in CSD. Then analyzed tasks failure ratio in both software production phase and identified task cycle.

This study identified four patterns for sequence of task arrival per project: 1) Prior Cycle, 2) Current Cycle, 3) Orbit Cycle and 4) Fresh Cycle. The empirical analysis support that the prior cycle led to the lowest task failure of 4% while fresh cycle resulted n the highest level of task failure(i.e 44%).

In future, we would like to use the identified task cycle and test them via crowdsourced task scheduling methods.

References

1. Alelyani, T., Mao, K., Yang, Y.: Context-centric pricing: early pricing models for software crowdsourcing tasks. In: Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering. pp. 63–72 (2017)
2. Archak, N.: 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 (2010)
3. Bernstein, M.S., Brandt, J., Miller, R.C., Karger, D.R.: Crowds in two seconds: Enabling realtime crowd-powered interfaces. In: Proceedings of the 24th annual ACM symposium on User interface software and technology. pp. 33–42 (2011)
4. Difallah, D.E., Demartini, G., Cudré-Mauroux, P.: Scheduling human intelligence tasks in multi-tenant crowd-powered systems. In: Proceedings of the 25th international conference on World Wide Web. pp. 855–865 (2016)
5. Faradani, S., Hartmann, B., Ipeirotis, P.G.: What's the right price? pricing tasks for finishing on time. In: Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence (2011)
6. Gordon, G.: A general purpose systems simulation program. In: Proceedings of the December 12-14, 1961, eastern joint computer conference: computers-key to total systems control. pp. 87–104 (1961)
7. Howe, J.: Crowdsourcing: How the power of the crowd is driving the future of business. Random House (2008)
8. Jiang, H., Matsubara, S.: Efficient task decomposition in crowdsourcing. In: International Conference on Principles and Practice of Multi-Agent Systems. pp. 65–73. Springer (2014)
9. Khanfor, A., Yang, Y., Vesonder, G., Ruhe, G., Messinger, D.: Failure prediction in crowdsourced software development. In: 2017 24th Asia-Pacific Software Engineering Conference (APSEC). pp. 495–504. IEEE (2017)
10. Khazankin, R., Psaier, H., Schall, D., Dustdar, S.: Qos-based task scheduling in crowdsourcing environments. In: International Conference on Service-Oriented Computing. pp. 297–311. Springer (2011)
11. Lakhani, K.R., Garvin, D.A., Lonstein, E.: Topcoder (a): Developing software through crowdsourcing. Harvard Business School General Management Unit Case (610-032) (2010)
12. LaToza, T.D., Van Der Hoek, A.: Crowdsourcing in software engineering: Models, motivations, and challenges. IEEE software **33**(1), 74–80 (2015)
13. Mao, K., Capra, L., Harman, M., Jia, Y.: A survey of the use of crowdsourcing in software engineering. Journal of Systems and Software **126**, 57–84 (2017)
14. Mao, K., Yang, Y., Li, M., Harman, M.: Pricing crowdsourcing-based software development tasks. In: 2013 35th International Conference on Software Engineering (ICSE). pp. 1205–1208. IEEE (2013)
15. Marcus, A., Wu, E., Karger, D., Madden, S., Miller, R.: Human-powered sorts and joins. arXiv preprint arXiv:1109.6881 (2011)
16. Mejorado, D.M., Saremi, R., Yang, Y., Ramirez-Marquez, J.E.: Study on patterns and effect of task diversity in software crowdsourcing. In: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 1–10 (2020)
17. Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L.: An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. Management science **44**(5), 714–729 (1998)

18. Ngo-The, A., Ruhe, G.: Optimized resource allocation for software release planning. IEEE Transactions on Software Engineering **35**(1), 109–123 (2008)
19. Reinertsen, D.G.: Celeritas publishing (2009)
20. Ruhe, G., Saliu, M.O.: The art and science of software release planning. IEEE software **22**(6), 47–53 (2005)
21. Saremi, R.: A hybrid simulation model for crowdsourced software development. In: Proceedings of the 5th International Workshop on Crowd Sourcing in Software Engineering. pp. 28–29 (2018)
22. Saremi, R., Lotfalian Saremi, M., Desai, P., Anzalone, R.: Is this the right time to post my task? an empirical analysis on a task similarity arrival in topcoder. In: Yamamoto, S., Mori, H. (eds.) Human Interface and the Management of Information. Interacting with Information. pp. 96–110. Springer International Publishing, Cham (2020)
23. Saremi, R., Yang, Y., Khanfor, A.: Ant colony optimization to reduce schedule acceleration in crowdsourcing software development. In: International Conference on Human-Computer Interaction. pp. 286–300. Springer (2019)
24. Saremi, R.L., Yang, Y., Ruhe, G., Messinger, D.: Leveraging crowdsourcing for team elasticity: an empirical evaluation at topcoder. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). pp. 103–112. IEEE (2017)
25. Saremi, R.L., Yang, Y.: Dynamic simulation of software workers and task completion. In: 2015 IEEE/ACM 2nd International Workshop on CrowdSourcing in Software Engineering. pp. 17–23. IEEE (2015)
26. Saremi, R.L., Yang, Y.: Empirical analysis on parallel tasks in crowdsourcing software development. In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW). pp. 28–34. IEEE (2015)
27. Stol, K.J., Fitzgerald, B.: Two's company, three's a crowd: a case study of crowdsourcing software development. In: Proceedings of the 36th International Conference on Software Engineering. pp. 187–198 (2014)
28. Surowiecki, J.: The wisdom of crowds. Anchor (2005)
29. Urbaczek, J., Saremi, R., Saremi, M.L., Togelius, J.: Scheduling tasks for software crowdsourcing platforms to reduce task failure. arXiv preprint arXiv:2006.01048 (2020)
30. Wang, H., Ren, Z., Li, X., Jiang, H.: Solving team making problem for crowdsourcing with evolutionary strategy. In: 2018 5th International Conference on Dependable Systems and Their Applications (DSA). pp. 65–74. IEEE (2018)
31. Yang, Y., Karim, M.R., Saremi, R., Ruhe, G.: Who should take this task? dynamic decision support for crowd workers. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–10 (2016)
32. Yang, Y., Saremi, R.: Award vs. worker behaviors in competitive crowdsourcing tasks. In: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 1–10. IEEE (2015)
33. Yue, T., Ali, S., Wang, S.: An evolutionary and automated virtual team making approach for crowdsourcing platforms. In: Crowdsourcing, pp. 113–130. Springer (2015)