Ask the Crowd: Scaffolding Coordination and Knowledge Sharing in Microtask Programming

Thomas D. LaToza[1, 2], Arturo Di Lecce[2], Fabio Ricci[2], W. Ben Towne[3], André van der Hoek[2]
[1] George Mason University, Fairfax, VA, USA, tlatoza@gmu.edu
[2] University of California, Irvine, Irvine, CA, USA, {arturodilecce, f.ricci89}@gmail.com; andre@ics.uci.edu
[3] Carnegie Mellon University, Pittsburgh, PA, USA, wbt@cs.cmu.edu

***Abstract*—Programming work is inherently interdependent, requiring developers to share and coordinate decisions that crosscut the structure of code. This is particularly challenging for programming in a microtasking context, in which developers are assumed to be transient and thus cannot rely on traditional learning and coordination mechanisms such as an extended onboarding process and code ownership. In this paper, we explore scaffolding coordination and knowledge sharing through a question and answer system, structuring project knowledge and coordination into questions and answers. To investigate its potential for enabling coordination in a microtask setting, we implemented a Q&A system for microtask programming work and conducted a user study where a crowd used it to coordinate their work on a software project over a 30-hour period. The results reveal both the potential for the use of Q&A systems for within-project coordination and challenges that this approach brings.**

Keywords—programming environments; crowdsourcing; microtasking; knowledge sharing; question answering

I. INTRODUCTION

As developers fix bugs and implement features, developers make decisions and seek to discover existing decisions that may inform and constrain their choices [18]. As architectural and even implementation decisions often span individual functions, classes, and modules, developers must gain a broader understanding of the code in order to understand how to successfully make changes without introducing new defects. To do so, developers use a wide variety of mechanisms: reading code, documentation, and design documents and coordinating using meetings, emails, IM, and issue trackers. To reduce what they must know, software development projects often employ code ownership, where developers become experts in modules over time through learning activities and specialization. Yet developers still find effective coordination and knowledge sharing to be a significant challenge [22], especially understanding past design decisions [18].

Microtask crowdsourcing envisions a radically different model of work [8]. Rather than working in long-term teams and on ongoing work items, workers are recruited using an open call and are assumed to be transient, working on short, self-contained microtasks without an extended onboarding process. As a result, microtask crowdsourcing has traditionally been used for embarassingly parallel tasks requiring little context such as labeling images [1] or transcribing video [14]. Compared to traditional work structures, microtask crowdsourcing offers several potential benefits, including large decreases in clock time through parallelism, a more fluid labor force, and collection of diverse ideas from the crowd.

More recently, work has begun to explore the use of microtask crowdsourcing in complex work requiring coordination and knowledge sharing [10][6][2][16][15][24]. In previous work, we introduced an approach for microtask programming [21]. In microtask programming, all work occurs through self-contained microtasks providing a local view of a single artifact – a function or test – rather than a global view of the entire codebase. For example, workers may be provided with a function description and be asked to list test cases or sketch part of an implementation, or be provided with a set of failing tests and a function and be asked to fix a bug.

A key challenge in microtask programming is coordination: how can a large crowd of transient workers working on small tasks coordinate their contributions into a coherent whole [19]? Observations of microtask programming in the small have found a need for workers to coordinate design decisions capturing crosscutting concerns across artifacts. Mechanisms such as global chat can enable real-time coordination amongst workers. However, as transient workers come and go and must identify and understand past decisions, threads with hundreds of responses bury ideas and are impenetrable to newcomers [17], creating contribution barriers that break the microtask model of short, self-contained work. Employing leaders to manage work and coordination diminishes the crowd's abilitity to make their own decisions and shape the final product. Thus, we sought to explore mechanisms for explicitly coordinating design decisions amongst transient workers. We designed and implemented an approach for scaffolding within-project coordination and knowledge sharing with a Q&A system, enabling workers to ask questions and discuss answers, connected directly to artifacts. To examine the potential of this approach, we conducted a study in which a crowd of 20 coordinated their work across a 30 hour period. The results reveal both the potential of Q&A for within-project coordination and some of the challenges this approach brings.

II. RELATED WORK

In microtask crowdsourcing, workers are traditionally treated as interchangeable automatons in a workflow, with limited context or awareness of the overall work product [8]. This approach can scale to surprisingly complex tasks, such as editing papers [4], coding videos [14], and multistep workflows [13][23]. But this model begins to break down as work becomes more complex and interdependent. Work has investigat-

This work was supported in part by the NSF under 1414197.

978-1-4673-7457-6/15/$31.00 ©2015 IEEE

ed extending this model by designing the workflow itself around identifying and processing interdependencies [6][2], providing a shared global coordination forum [16], letting workers see the current state of the work product [15], or abandoning the notion of transient workers who may disappear in favor of short-term commitments within ad-hoc teams [24]. Crowdsourcing systems for programming have often relied on a singular leader to coordinate and manage work (e.g., co-pilots in TopCoder[1] and "original programmers" in micro-outsourcing [10][9]).

Much work has examined coordination and knowledge sharing practices and challenges in traditional software development teams and open source projects (e.g., [7][5][3][22][20][11]). Microtask crowdsourcing differs in several important ways, particularly that workers are transient and may come and go freely and that parallelizing work is a key goal. As a consequence, traditional crowdsourcing coordination models emphasizing leadership (e.g., the onion model [11] or even leadership that is partially distributed [17]) break down, as the system cannot assume that leaders will remain involved or be online when needed. Thus, our key design goal was to enable coordination in this context, moving beyond both global chat and a central leader to help the crowd itself organize information and reach consensus.

III. Background - Microtask Programming

In previous work, we introduced a microtask approach to programming [21], which we briefly review here. In microtask programming, transient workers contribute to a software project through short, self-contained microtasks, each describing a specific task to be completed on a single function or test. Each microtask is designed to be self-contained, providing instructions, relevant context, and an artifact editor, enabling workers to contribute without an extended onboarding process. All microtasks are automatically generated and assigned. Microtasks include editing function descriptions and implementations, debugging, editing test cases and tests, reviewing completed microtasks, and debugging. To prevent individual workers from blocking progress by working on a microtask for an extended period of time, all microtasks have a ten minute time limit. Several coordination mechanisms are offerred. Implementing a function is iterative: workers can partially implement a function, using pseudocode to describe further work to be done, which then generates a subsequent microtask for a worker to continue its implementation. Coordination also occurs across dependencies between artifacts. For example, workers may, if needed, change the signature of a function, creating microtasks to update call sites and tests, and workers implementing a test may report an issue with the corresponding function description, generating a microtask on the function to resolve the issue and edit the function. Finally, completed microtasks are reviewed by another worker, and may be accepted as is or reissued, creating a microtask for the artifact to be revised. Microtask programming has been implemented in CrowdCode, a web-based IDE.

[1] www.topcoder.com

IV. System Design

We set out to enable coordination and knowledge sharing among transient microtask workers. Inspired by the design of knowledge sharing systems such as StackOverflow, coordination is explicitly structured as questions and answers. Workers may, at any time, ask and answer questions, tag discussions, and comment, up-vote, and down-vote responses. In contrast to their traditional use for global knowledge sharing across projects, our Q&A system was designed for coordination and knowledge sharing within a project. Thus, discussions are deeply integrated with specific parts of the code, offering workers the opportunity to collaboratively discuss specific decisions within a project, contextualized with its artifacts. In contrast with both Q&A systems and issue trackers, questions are not owned and managed by a requestor. Rather, the crowd itself is empowered to freely edit questions and mark discussions as closed where appropriate. In the following sections, we discuss the design in detail.

A. Finding Answers

When a worker has a question, they may first check if the crowd has already answered it. The top-level view of the Q&A system lists all questions (Figure 1, left), with each question's title, a summary of its activity, its tags, and whether it is currently open (yellow background) or closed (gray background). To enable workers to see when a question has recent activity they have not yet viewed, new answers and comments that the worker has not seen are also listed. Workers may browse the list of questions or search using a full text search or for questions with specific tags. Clicking on a question opens its discussion page (Figure 1, right).

The list of questions is divided into two sections. The top section lists questions that have been marked as relevant to the current function. In this way, workers can immediately see discussion of questions relevant to their current work. The bottom section lists all other questions (relationships of these questions to other functions, if any, are hidden). If a worker discovers a question that seems relevant to their current work, they can click a toggle button to mark (or unmark) it as related to the current artifact.

B. Asking Questions

While working on a microtask, a worker may ask a question, providing a title and description and optionally tags describing the question. Workers may additionally choose to mark the question as related to the function in which they are currently working. When a new question is asked, workers

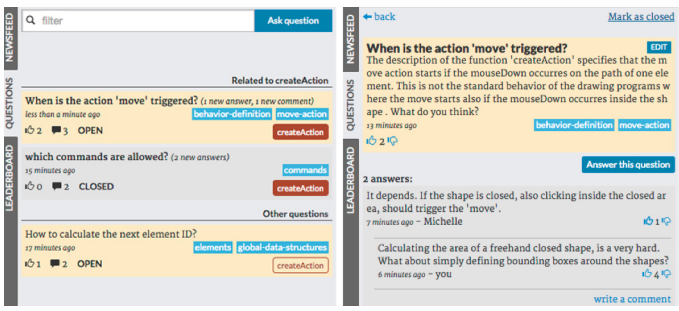

Fig. 1. Panels for browsing questions (left) and discussing answers (right).

receive an immediate notification.

C. Answering and Discussing Questions

To enable open contributions by the crowd, any worker may, at any time, discuss a question. Discussion for each question is threaded into answers and comments on answers (Figure 1, right). Workers may choose to indicate their agreement or disagreement with a question, answer, or comment through an up-vote or down-vote. Each question, answer, and comment is listed with its net up-votes. Alternatives and discussions of alternatives are in this way made explicit and recorded, enabling workers to directly see the rationale and discussion behind decisions.

Workers may use question threads both synchronously and asynchronously. To support the use of rapid synchronous discussion to generate and discuss alternatives, all Q&A content is immediately synchronized with all workers. Workers may carry on a real-time conversation, allowing, for example, a worker to answer a question, see a comment raising an issue, and post an additional comment to address the issue in response. To support asynchronous awareness of updates to discussions, workers automatically follow a discussion after participating by contributing a question, answer, or comment. Whenever a contribution is made, all workers following the discussion receive a notification. If a worker is logged out, the notification is deferred.

One model of incorporating Q&A into microtasking work would be for question answering or discussion to itself be a microtask. However, such a model has significant drawbacks. Workers may vary significantly in their knowledge and ability to answer a question. Soliciting answers from the whole crowd enables workers to self-select their ability and interest to contribute. Second, organizing contributions as microtasks delays responses, as there is a lag between when the microtask is created and when the microtask is assigned and completed. This may discourage discussion and debate of alternatives, as workers cannot engage in a single synchronous disucssion.

Due to these considerations, we chose to enable Q&A contributions to be made orthogonal to microtask work. The Q&A system is displayed as a tab within a side panel adjacent to the microtask work area. Workers working on a microtask can follow discussion on a question while working, choosing to participate as they wish. As workers may be transient and leave, we do not assume that a question asker will return to mark a question as answered. Instead, each question is labeled as Open or Closed, and any worker may toggle its status at any time. Additionally, the question title, text, and tags are collaboratively editable, allowing any worker to, at any time, clarify the question based on the ongoing discussion. Interactions with the Q&A system are gamified, incentivizing participation and consistent with CrowdCode. Workers receive 3 points when their question is up-voted, 2 points when an answer is up-voted, and 1 point when a comment is up-voted.

V. User Study

To examine the prospects and challenges of scaffolding coordination and knowledge sharing, we performed a user study. Rather than study the use of a Q&A system in a controlled but artificial setting through a controlled experiment, we chose to instead conduct a larger single study, enabling observation of the social dynamics that arose as a crowd worked together. 20 participants were recruited from graduate and undergraduate students at several universities and professional software developers through personal contacts. Participants were geographically distributed. Participants had a mean of 2.7 years of professional experience. 13 had used a previous version of CrowdCode without the Q&A system. To simulate a setting in which transient workers may freely join and leave, participants were invited to participate whenever they wished across a 30 hour period for up to 4 hours. Participants worked to implement the logic of a simple spreadsheet application. Participants were paid $20 an hour, based on their time interacting with CrowdCode. To discourage gaming the system, pay was not tied to points accrued in CrowdCode. Participants' work was captured through a screencast and event logging within CrowdCode, and participants completed a survey on their experiences.

VI. Results

Overall, participants interacted with CrowdCode for a total of 79 hours during the 30 hour study period. Of this time, workers spent 52 hours working on microtasks, during which they completed 684 microtasks, implemented 20 functions and 62 unit tests, and wrote 353 lines of function code and 2202 lines of test code. Participants spent the largest portion of their time working on Debug Test Failure, Review, and Edit Function microtasks. Workers were transient, logging into the environment, on average, a total of 2.0 times for an average time of 121 minutes. Workers interacted with CrowdCode for an average of 3 hours and 52 minutes each. Due to worker's varied time zones, during 80% of the study period at least one worker was active in CrowdCode, with as many as 7 workers concurrently active and 2.3 on average.

A. Use of the Q&A System

Workers made use of the Q&A system to coordinate and share knowledge. For example, after reading through several questions and answers on the structure of spreadsheet formulas, P13 asked a question seeking a common definition, adding the tag "definitions." He later edited the question to add additional descriptive tags, including "formula" and "function." P7, while working on a test case for the function *computeFunct-ForRange*, went through the open questions to find information about the function. He found P13's question and up-voted it. After gaining additional knowledge about formulas, he answered it, proposing two alternatives. P12, noticing the question, then gave his opinion on one of the issues P7 raised.

Overall, workers asked a total of 26 questions. Questions focused on a variety of topics. Some focused on crosscutting decisions, such as how insertions and deletions would be handled or what spreadsheet functions should be supported. Others sought to clarify particular aspects of function descriptions. In other cases, "questions" were appropriated to state decisions that workers believed had already been made, indiciating to others a convention to be followed. Workers also used the Q&A system to ask questions about how CrowdCode itself worked. Finally, workers sometimes used Q&A for explicit coordination, requesting others who might be working

on an artifact to take an action with it or to gain an understanding of the current overall status of the project.

All questions received at least one answer, and all questions were marked as closed by the crowd by the end of the study. However, in many cases, questions were left unanswered for significant periods of time. 25% of questions were answered by another worker within 5 minutes, 42% within an hour, and 85% in the first 10 hours. As a result, workers sometimes simply made a decision themselves, sometimes recorded as an answer to their own question. Answers were delayed for several reasons. At many points, few were available to answer questions. Moreover, one reported that, when working on particularly challenging microtasks, the 10 minute microtask time limit meant that they had no time to answer questions. Questions varied in the amount of discussion they generated and the attention they received. Questions received on average 2.6 answers and comments and 30 views; some questions received as many as 6 answers and comments and 35 views.

Participants sometimes checked artifacts for conformance to the decisions they had seen in the Q&A system, reporting issues when the artifacts did not conform. For example, when reviewing a test case, P12 reported that the test and description of a function did not conform to a decision, citing a question title in explaining his review. Similarly, participants sometimes used the issue reporting mechanism to report non-conforming artifacts, justifying their issues with a citation to the decision.

In other cases, participants still made conflicting decisions, despite the Q&A system, as they did not realize that the decisions they were making created conflicts. In CrowdCode, work on new functions proceeds along two parallel paths. Beginning with a function description, parallel workflows separately implement the function and its tests. The workflows converge when the tests are run, which may then generate a Debug Test Failure microtask. Workers separately implementing the function and each test often made widely varying decisions about the interpretation of the function description. Rather than use the Q&A system to state or discuss these interpretations, workers often simply made decisions. Because these conflicts were only caught after code and tests had already been written, resolving and adjudicating these conflicts through the Debug microtask often required signficant rework.

B. Perceptions of the Effectiveness of the Q&A System

Participants reported varied opinions about the efficacy of using a Q&A system for coordinating and sharing knowledge in a microtasking context. Compared to global chat, several felt that Q&A more effectively supported knowledge sharing. Participants cited a range of advantages: that it was "*easier to transmit and spread the information*" (P2), easier to "*find relevant issues very easily*" (P5), better than "*tutorials and documentation*" (P18), less distracting, easier to reach agreement, and easier to get up to speed. Others disagreed, preferring the global chat they had used in a previous version of CrowdCode. These participants felt that "*time delay is one of the most important issues*" (P14), reflecting frustration that their questions were not quickly answered and a belief that questions would be answered more quickly in global chat. This may have been influenced by the experiences of some workers who had used global chat in a previous version of CrowdCode; however, in contrast to this study and more realistic microtask settings, workers were not transient and all participated in a single synchronous session, ensuring there many were always available to quickly answer questions.

Participants expressed a wide range of ideas for improving coordination in a microtask work context. One wished that "*If you begin a task related to function F and there have been new discussion posts since you last did something related to F, then you should be forced to see the recent discussion posts*" (P7). Several wished there to be person to person communication to discus reviews or continue work begun by another worker. P16 expressed the desire for workers working on related tasks – implementing tests for the same function – to have the ability to collaborate in real time. Another proposed that there be a "*project owner/manager… that could benevolent[ly] dictate his opinion*" (P9). Several expressed alternative ideas for structuring information, including a wiki for terms and definitions (P2, P3, P13, P15), adding memos to the Q&A system (P1), and better sorting and organization of questions.

VII. DISCUSSION

Enabling effective coordination among transient microtask workers is a key challenge in enabling microtasking to be used for more complex, interdependent activities such as programming. Our results revealed that a crowd of transient workers completing highly interdependent programming microtasks was able to use a Q&A system to help coordinate their work. Rather than a leader or client dictating decisions, workers themselves had a voice in shaping the direction of their work [12]. The results also highlighted several additional conditions for a Q&A system to effectively support crowd work, which the design only partially supported. Workers need to be made aware of when their work might create conflicts and, in such situations, incentivized to ask questions so that conflicts are identified and resolved earlier. As workers ask questions, they may create related and duplicative questions, highlighting the importance of explicitly identifying and merging or connecting related questions to keep information well-organized. It is important for decisions to be deeply and prominently integrated with the artifacts themselves, so that relevant decisions are made apparent and surfaced as workers begin working on each microtask. Finally, in order for questions to be answered in a timely fashion, it is important to consider the size and work times of the crowd, so that workers can be online and available at similar times.

More broadly, our results demonstrate the potential for capturing software project knowledge at a more fine-grained level. Software projects traditonally capture knowledge in issue trackers, design documents, wikis, and emails. Yet these artifacts are ill-suited for more low-level programming decisions that shape programming tasks but still crosscut the structure of software. Our findings suggest that Q&A systems might be valuable within software projects, helping developers to more explicitly coordinate important decisions in ways that externalize decisions and capture rationale.

ACKNOWLEDGMENT

We thank the participants for their time.

REFERENCES

[1] L. von Ahn and L. Dabbish, "Labeling images with a computer game. *Conference on Human Factors in Computing Systems* (CHI), 2004, pp. 319-326.

[2] P. André, A. Kittur, and S. P. Dow, "Crowd synthesis: extracting categories and clusters from complex data." *Conference on Computer Supported Cooperative Work* (CSCW), 2014, pp. 989-998.

[3] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: discovering and exploiting relationships in software repositories." *International Conference on Software Engineering* (ICSE), 2010, pp. 125-134.

[4] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich, "Soylent: a word processor with a crowd inside." *Symposium on User interface software and technology* (UIST), 2010, pp. 313-322.

[5] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures." *Trans. Software Eng.,* 35(6), 2009, pp. 864-878.

[6] L. B. Chilton, G. Little, D. Edge, D. S. Weld, and J. A. Landay, "Cascade: crowdsourcing taxonomy creation." *Conference on Human Factors in Computing Systems* (CHI), 2013, pp. 1999-2008.

[7] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems. " *Commun. ACM,* 31 (11), November 1988, pp. 1268-1287.

[8] A. Doan, R. Ramakrishnan, and A. Y. Halevy, "Crowdsourcing systems on the World-Wide Web." *Commun. ACM,* vol. 54 (4), April 2011, pp. 86-96.

[9] M. Goldman, "Software development with real-time collaborative coding in a Web IDE." *Symposium on User Interface Systems and Technologies (UIST)*, 2011, pp. 155-164.

[10] M. Goldman, G. Little, and R. C. Miller, "Collabode: collaborative coding in the browser." *Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2011, pp. 155-164.

[11] C. Jensen and W. Scacchi, "Role migration and advancement processes in OSSD projects: a comparative case study." *International Conference on Software Engineering (ICSE)*, 2007, pp. 364-374.

[12] A. Kittur, J. V. Nickerson, M. Bernstein, E. Gerber, A. Shaw, J. Zimmerman, M. Lease, and J. Horton, "The future of crowd work." *Conference on Computer Supported Cooperative Work (CSCW)*, 2013, pp. 1301-1318.

[13] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, "CrowdForge: crowdsourcing complex work." *Symposium on User interface software and technology* (UIST), 2011, pp. 43-52.

[14] W. S. Lasecki, M. Gordon, D. Koutra, M. F. Jung, S. P. Dow, and J. P. Bigham, "Glance: rapidly coding behavioral video with the crowd." *Symposium on User interface software and technology* (UIST), 2014, pp. 551-562.

[15] W. S. Lasecki, J. Kim, N. Rafter, O. Sen, J.P. Bigham, M.S. Bernstein. "Apparition: crowdsourced user interfaces that come to life as you sketch them." *Conference on Human Factors in Computing Systems* (CHI), 2015.

[16] W. S. Lasecki, R. Wesley, J. Nichols, A. Kulkarni, J. F. Allen, and J. P. Bigham, "Chorus: a crowd-powered conversational assistant." *Symposium on User Interface Software and Technology* (UIST), 2013, pp. 151-162.

[17] K. Luther, C. Fiesler, and A. Bruckman, "Redistributing leadership in online creative collaboration." *Conference on Computer Supported Cooperative Work* (CSCW), 2013, pp. 1007-1022.

[18] T. D. LaToza, D. Garlan, J. D. Herbsleb, B. A. Myers, "Program comprehension as fact finding." *European Software Engineering Conference and Foundations of Software Engineering (ESEC/FSE)*, 2007, pp. 361-370.

[19] T. D. LaToza and A. van der Hoek, "A vision of crowd development". *International Conference on Software Engineering, NIER track (ICSE NIER), 2015.*

[20] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code." *Workshop on the Evaluation and Usability of Programming Languages and Tools* (PLATEAU), 2010, 6 pages.

[21] T. D. LaToza, W. B. Towne, C. M. Adriano, and A. van der Hoek, "Microtask programming: building software with a crowd." *Symposium on User Interface Systems and Technology (UIST),* 2014, pp. 43-54.

[22] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits." *International Conference on Software Engineering (ICSE)*, 2006, pp. 492-501.

[23] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, "TurKit: human computation algorithms on mechanical turk." *Symposium on User interface software and technology* (UIST), 2010, pp. 57-66.

[24] D. Retelny, S. Robaszkiewicz, A. To, W. S. Lasecki, J. Patel, N. Rahmati, T. Doshi, M. Valentine, and M. S. Bernstein, "Expert crowdsourcing with flash teams." *Symposium on User Interface Software and Technology* (UIST), 2014, pp. 75-85.