Labeling relevant skills in tasks: can the crowd help?

Rafael Leano[1], Zhendong Wang[2], Anita Sarma[1,2]

[1]School of EECS, Oregon State University
Corvallis, OR, 97331
{leanor, anita.sarma}@oregonstate.edu

[2]CSE Department, University of Nebraska, Lincoln
Lincoln, NE, 68588
zwang@cse.unl.edu

Abstract— **Identifying the skills required to complete a task is an important aspect of software development. Managers perform this when triaging and assigning tasks. Developers do it when picking tasks to work on. However, this endeavor is time-consuming for experts, and difficult for newcomers. In this paper, we investigate a crowd-based approach to find the skillset of a task. Our results indicate that the crowd is able to identify the skills with a recall of 0.67 and a precision of 0.76. Further, we found no difference between novice and experienced crowd workers in identifying skills. Finally, our results suggest that tasks should be reviewed by at least four workers to leverage voting at a 25% agreement.**

Keywords— crowdsourcing, skill, task context

I. Introduction

Understanding the skills required to complete a task is an important aspect of software development, where skills mean the capability of performing an activity [1]. In software development, skills can be knowledge about programming concepts (e.g., event handling), or about an API, a library, or a class.

Managers (or developers) need to identify the skill requirements for a task either to identify who can work on a task (triage it), or to identify tasks that are appropriate for newcomers (onboarding). Project managers manually triage or assign tasks by using their experience and knowledge about the project [2] However, such manual labor is tedious and effort intensive [3]; and not scalable for large projects like Mozilla, where 300 bugs need triaging on a daily basis. Similarly, the number of issues in the Eclipse IDE became so numerous that it moved to a decentralized model, where each component team monitors and triages bugs. Therefore, mechanisms that help with triaging can free up developer resources [4].

On the other hand, in open source (OSS) projects, developers identify the tasks that they can implement [5]. Newcomers are encouraged to find tasks that are suited to them, and get their "hands dirty" to get onboarded into the project [6]. However, when new contributors do not have experience with the project structure, it is difficult for them to identify the skills a task requires and whether they can contribute to it [7]. In fact, Steinmacher et al. [8] report one of the main barriers to newcomers is: "*how to make the newcomers aware of the skills needed; and, how to support them choosing the right tasks that fits with their knowledge*?".

To the best of our knowledge, no one has yet investigated how to identify the skills required for a task. Research has investigated triaging problems by focusing on expert identification [3], [9]. They use data mining and information retrieval techniques to identify past similar tasks to predict which files are relevant for a current bug fix [10], or use past edits to a file to find experts [11]. Research on helping newcomers to facilitate onboarding has, thus far, focused on explaining the development and joining processes [5], or classifying the barriers that newcomers face [7].

In this paper, we investigate an alternate mechanism to manual identification of skillset requirements by experts or individual newcomers. The problem that we aim to solve is that while experts are capable of identifying skills, their time is limited. On the other hand, individual newcomers lack the expertise to perform this step.

Crowdsourcing has been powerful in scientific domains (e.g., FoldIt [11] for solving protein-folding problem), and in software design and development (e.g., TopCoder, Kaggle), among others [12]. Research in software engineering has started to explore the barriers that exist and the infrastructure that is needed to enable programming tasks to be crowd-sourced [13].

We, therefore, investigate whether crowd-sourcing can help in skill labeling. We answer the following research questions:

RQ1. How well can a crowd identify the relevant skillset of a task from its context?

We performed our study in oDesk, which is one of the largest online crowd community with skilled workers. In the study, participants found the skills required for a task (i.e., *labeled* the task) of an open source project (*AndroidSwipeLayout*). This labeling required reviewing the task description, issue discussion, and the associated source code. Our results show that the crowd is able to identify most of the relevant skills required for the task (0.67 recall), with a best precision of 0.76.

RQ2. Does skill identification (labeling) require an experienced crowd? We further analyzed the experience background of the crowd workers, and found that experience (tenure in software development, project role) did not impact the results.

RQ3. How many workers should label a task? Using more workers to label a task increases the precision of the skills, but assigning too many workers is inefficient. Our results suggest that four workers can get good results.

In summary, the contributions of our work are:

- We show the feasibility of finding relevant skills for a task using a crowd, where they are unfamiliar with the project.
- We find that novice developers can find skills as well as experienced developers.
- In our context, the crowd results can be enhanced by voting: selecting labels that meet at least 25% agreement, and the best results occur with four or more participants.

II. Background

In the context of software development, a skill can represent knowledge about certain programming concepts or represent knowledge at a finer grain (even specific class names). In our study, we broadly classify skills related to *general programming* (e.g., inheritance and event handling), *language constructs* (e.g., Java Annotations, Java Collections), or *project-specific* concepts (e.g., classes, Maven, or other APIs).

Work is partially supported by NSF: IIS-1314365 and CCF-1253786.

978-1-5090-0252-8/16/$31.00 ©2016

A. Crowdsourcing

Crowdsourcing leverages the large community of online users with different talents and views to achieve a task. The variety of workers enriches the types of tasks and solutions obtained from a single expert [14]. Crowdsourcing works by decomposing large, complex tasks into smaller jobs that workers can perform in a single short session. So far, the crowd has been used successfully in experiments like FoldIt [11], and PipeJam [15]. Research in computer science has used the crowd for studies that need surveys, data cleaning, usability testing, annotating, etc. [14], [16]–[18].

However, one challenge of crowdsourcing is the quality of the results [19]. It is important to ensure that participants have the requirements for the job, that they do not game the system, and that they follow the instructions correctly [14], [20]. We ameliorate these issues by posting the job in oDesk, which uses certification and rating systems that are posted on profiles.

Finally, having different workers repeat the same job allows filtering results to increase their quality. The answers can be aggregated and filtered based on the characteristics of the task [21]. One such strategy is majority voting, where the results with an agreement above a given threshold are retained.

B. Related Work

Many studies have developed approaches to find, for a given task, (1) the best suited developer (i.e., expert) or (2) similar tasks. For the former, MacDonald et al. [22] have artifacts vote for their experts based on development history. Similarly, DebugAdvisor [23] identifies the relevant files and documentation for the task and then searches for their experts. Finally, Anvik et. al [4] use machine learning on past tasks to find similar ones and recommend the developer that solved them. Our approach has a different goal; it offers a method for finding the relevant skills for new tasks. This information can prove useful for (1) managers matching the skills of the task with a developer, or (2) newcomers looking for a task that they can implement with their current skills.

III. APPROACH

Out study asked participants to find the skills required to complete a task by reviewing its *task context*, and writing down the skills found as labels. We call this process *labeling* a task. We measure the performance of the crowd using the common *recall (R), precision (P),* and *F-measure* metrics.

A. Study Design

For this study, the *task context* refers to the details about an issue or bug that can be found in the issue tracker or version control system: title, task description, issue discussions, and the associated source code (relevant files).

Project Selection: We required a project with public source code that linked their tasks (issue tracker) with the source code (versioning system). Additionally, as this was a first study we chose a relatively small project to ensure we could easily crowdsource the (skill labeling) job. Finally, we needed collaboration from the project owner to obtain the ground truth labels (the real, relevant skills required for a task in this project). The *AndroidSwipeLayout* project, an android swipe component, met all these requirements.

This project uses GitHub for versioning code and for issue tracking (hence there is issue-code linkage). It consists of four Java classes across two files of 1.5KLOC, and is well documented. The project owner (and main developer) labeled the task, which was our gold standard. He also reviewed the crowd generated labels and gave feedback about their quality.

Study Task: We focused on closed issues to ensure it was fixable and to know the files involved in the fix. It also provided us the exact LOC that were modified for the bug fix. On average, bug fixes included small changes (about 10 LOC) to the code base. The small size of the bug fixes (and the code base) meant that we did not need to break down the bug fix into multiple jobs, which could add noise to the study [13].

We identified about fifty bugs that match our criteria, from which we randomly selected one. The selected issue was bug #9, which concerned the drag action not working properly when it was over a swipe control (e.g., a volume bar), as the dragging got assigned to the latter. The issue was solved in commit (#93793031).

B. Crowd Source Study

In the study, participants were provided with the task context, and they had to identify the skills that they thought were needed to perform the bug-fix.

Crowd Environment: We chose the oDesk crowd platform because it provided more control over specifying the types of participants who could apply to the study. This helped us ensure that participants had a sufficient programming background (a common problem in crowdsourcing [14]).

oDesk offers certifications and quizzes for programming, such as *Java basics* or *Android programming*. Since our project uses Java, we asked workers to either have Java certification, or 60% or greater on their Java Quiz.

We also collected participants' demographic information: years of experience as a developer, Android experience, and their role in their last/current software project.

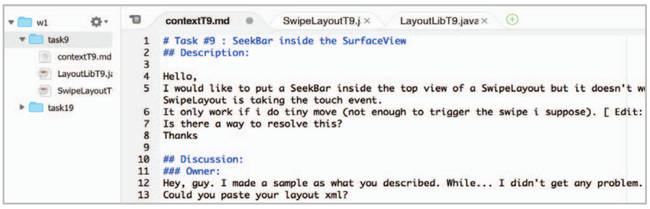
Participants: Our study stipulated a one-hour maximum duration with a compensation of $10. We received 18 applications. We accepted ten participants as some participants did not complete the initial demographic survey, scored less than 60% on the Java quiz, demanded a higher payout, or applied twice. Out of the ten accepted participants, one did not finish the study, and another did not correctly follow instructions. The remaining eight participants were classified as either *experienced* or *novices* according to their demographic information.

*P*articipants with 5 or more years as a developer, or with a senior role in their regular job were considered *experienced*; rest were considered *novices*. We had four participants in each category (Table IV), which helped us investigate the effects of development experience on the crowd results.

Crowd Workflow: Participants started by watching an 8-minute video about the environment and labeling process.

The environment was a modified version of the Cloud9 IDE (C9) as the editor (see Fig. 1). It allowed workers to log in, navigate through the files of the context (title and description, discussion, source code). We also logged their actions using the Cryolite [24] logger to get timestamps, the files and context elements viewed, and the time spent on each context element.

Participants used Google spreadsheets to record the skills as labels. They also linked the skills and the context where they



```
# Task #9 : SeekBar inside the SurfaceView
## Description:

Hello,
I would like to put a SeekBar inside the top view of a SwipeLayout but it doesn't w
SwipeLayout is taking the touch event.
It only work if i do tiny move (not enough to trigger the swipe i suppose). [ Edit:
Is there a way to resolve this?
Thanks

## Discussion:
### Owner:
Hey, guy. I made a sample as what you described. While... I didn't get any problem.
Could you paste your layout xml?
```

Fig. 1. Task labeling process with the task context in the C9 IDE.

found the skill. As each skill is represented by a label, henceforth, we use *labels* to represent the skills required to do a task.

In an exit survey participants rated the difficulty of the labeling process and the usefulness of tasks labeled with skills.

TABLE I. CROWD LABELS (IN GREY) VS. GOLD STANDARD

Labels	*Brief description*	*Labels*	*Brief description*
Adapter	Android concept	List	Programming concept
Android	Platform	Maven	Programming tool
Condition	Programming concept	Method Overload	Programming concept
DialogFragment	Android class	ObjectOriented	Programming concept
Event Handling	Android concept	Recursion	Programming concept
ViewDragHelper	Android class	SeekBar	Android class
GestureDetector	Android class	support.v4	Android package
GUI Layout	Android concept	SwipeDetector	Android class
HeadViewListAd.	Android concept	SwipeLayout	Project class
Heap	Programming concept	SwipeListener	Project class
Inheritance	Language concept	View	Android concept
Inner Classes	Language concept	XML	UI Layout in Android

C. Labels – Ground Truth and Aggregation

Gold Standard: The owner of the project labeled the tasks that he considered important for the bug fix. He identified a set of 24 labels (See Table I), and we use them to classify the labels from the crowd into: *true positives* (in the gold standard, shaded in grey), *false positives* (not in the gold standard), and *false negatives* (missed by the crowd).

Label Aggregation: We found the crowd labels often referred to the same or a very similar concept. For example, *user interface, GUI layout*, *Java GUI*, all refer to the same concept, but participants used slightly different labels. This is a common issue in labeling due to different spellings, synonyms, etc. In such cases, the labels need to be aggregated. In StackOverflow, for example, users with high reputation suggest tag synonyms, which are then validated by the community.

To address this issue, we perform label aggregation. Two of the authors individually examined all the labels, then clustered them into bag of equivalent words. Finally, only those labels that both researchers considered equivalent were aggregated. Table II shows a sample of the individual labels aggregated together in a bag.

TABLE II. SAMPLE OF AGGREGATED LABELS

Aggregated label	*Individual Labels in a bag*
Event Handling	Event Handling, onTouchEvent, ClickEvent, MotionEvent
GUI Layout	User Interface, GUI Layout, Java GUI
Map	Hashmap, Map

IV. RESULTS

After aggregating the labels from by the crowd, we get 29 unique *crowd* generated labels. Participants took an average of 33 minutes to label the task, most of which involved scanning the source code.

Our gold standard comprises 24 labels. Comparing the two sets (Fig. 2a) shows that the crowd found 16 *relevant* labels out of the 24: that is a recall of 0.67, a precision of 0.55, and an F-measure of 0.60 (See Table III, row 1). Therefore, we posit that the crowd can identify a majority of the skillset required for a task with good precision.

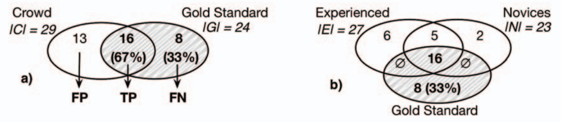


Fig. 2. (a) Labels found by the crowd and the gold standard, with the interpretation of each subset. (b) Labels from the experienced and novice users

Next, we investigate the labels from the crowd and their usefulness. Labels from the non-source part of the context (*ns-context*): title, task description, and discussions; had a higher number of true positives (83%), than labels from the source code (61%). Therefore, the ns-context provides a rich source of labels, and should be considered for labeling. Moreover, source code files may not differ between tasks, thereby generating the same labels, whereas the ns-context is unique per task.

We noted three things when exploring the relevant crowd labels (Table I). First, participants label class names or class attributes that are used frequently in the source code (e.g., the *SwipeListener* method), or are explicitly mentioned in the ns-context (e.g., *SeekBar*), whereas infrequent classes are ignored. In our case, the crowd ignored relevant classes for this reason. Second, classes that are similar (e.g., *MotionEvent*, *Touch Event*), or in the same package are selected as labels, even if their individual occurrences are low. Finally, programming skills (e.g. event handling or OO-concepts) were identified when *keywords* denoting such concepts (e.g., *implements*) appear in the source code.

For missing labels (false negatives), we find that: First, higher-level programming concepts (e.g., recursion, heap) are missed. This is likely because these are not evident from the source code and require an understanding of the program logic. Unsurprisingly, skills that are not mentioned are missed, for example XML. However, the owner of the project, mentioned that knowing XML is important for the bug fix, since, "*the (android) interface… depends on xml, especially layout, so, people need to know*". In a similar fashion, the inner classes or other attributes that were not visible were missed.

Observation: The crowd retrieves labels for concepts that are explicitly stated, or occur frequently in the code. Programming concepts that can be inferred from keywords are retrieved, but those that need to be inferred from program logic are easily missed.

In the non-relevant (false positives) labels, only one (*Swipe Denier*) came from the ns-context (discussion), the rest were from the code. This label was considered unimportant by the owner, "*SwipeDenier is being mentioned…it is only a suggestion [in the discussion]"*. We, thus, posit that discussions from non-core members (who may lack infrastructure experience), or that are about brainstorming are ill suited for skill labeling.

The other false positives came from the source code, these may be relevant for the particular file (or class), but not for the specific task: "*…yes for this [file], but not important here for this [fix]*". This suggests that the precision of the crowd results can be increased if we identify the specific parts of the code that are likely to change. This can be done using techniques such as fault localization [25], [26], or data mining [27].

Majority Voting: We next used majority voting [21] to enhance the quality of our results, as the crowd would agree (more) on relevant labels. Table III shows the results at different threshold levels. At 0% our results are the same from before (good recall, but lower precision). We see that higher agreement levels can lead to better precision, but lower recall. However, if the filtering is too aggressive then we retrieve fewer labels and precision is also affected. In our case, filtering at 25% had the highest precision (0.76) and recall (0.67).

Observation: A 25% agreement provides optimal results (Table III, row 2), as it provides a good trade-off between the number of participants needed and the quality of the results.

TABLE III. PERFORMANCE OF THE CROWD AT DIFFERENT FILTERING

row	*Filtering*	*True Positives*	*Retrieved*	*Recall*	*Precision*	*F1*
1	*None*	16	29	0.67	0.55	0.60
2	***25%***	**16**	**21**	**0.67**	**0.76**	**0.71**
3	*40%*	12	17	0.50	0.71	0.58
4	*50%*	7	10	0.29	0.70	0.41

Next, we answer the remaining two research questions: (1) what is the effect of experience, and (2) how many participants are needed to get optimum results.

1) Experienced vs. Novices

We compared the labels obtained by both novices and experienced developers. However, we did not see a difference in the number of relevant labels (Fig. 2b): both groups identified 16 of the 24 relevant labels (Recall (R) = 0.67, see Table IV). Notice that the crowd as a whole (Table III) performs better than individual participants (Table IV).

Novices had fewer non-relevant labels, resulting in a higher precision (P: 0.70 vs. 0.59 in Table IV). But they also found fewer labels: 12.5 on average vs. 14.2 for experienced developers. However, one experienced worker (P3) provided 21 labels, deviating from all other participants. Further, all the (6) non-agreements came from this participant. Thus, if he is considered as an outlier and removed, the precision rises to 0.74, and recall drops to 0.58. This brings the results of the novices and experienced developers even closer to each other.

Observation: Programming experience *does not* matter for identifying the relevant skills in a task in an unfamiliar project.

2) Sensitivity analysis of the number of participants

We randomly ordered the participants and added their results one by one to obtain the cumulative metrics (recall, precision, and F1). We plot the behavior of each metric in Fig. 3.

Solid lines show cumulative results for each additional participant (without filtering); dotted lines show filtering at 25%. Non-filtered data show that more workers increases recall (more labels), until it plateaus at four participants. The filtered data does not suffer from this issue as the voting controls the variance. We see that in the filtered case, the results get progressively better with each worker, with slight increases.

Observation: Tasks should be labeled by at least 4 participants to leverage voting at 25% agreement. While additional participants are useful, they only bring marginal improvement.

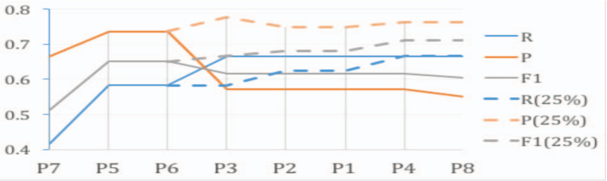


Fig. 3 Cumulative recall, and F1 per participant

In summary, we find that a crowd of developers can be effective in labeling the skills required for a task, even if they are unfamiliar with the project (RQ1). We learned that development experience is not needed to label tasks (RQ2). Finally, four workers provide enough repetition and diverse labels to benefit from a 25% agreement threshold (RQ3).

TABLE IV. PERFORMANCE METRICS OF INDIVIDUAL WORKERS

	R	**P**	**F1**		**R**	**P**	**F1**
Novice	**0.67**	**0.70**	**0.68**	**Experienced**	**0.67**	**0.59**	**0.63**
P1	0.37	0.82	0.51	P3	0.42	0.48	0.44
P2	0.29	0.70	0.41	P4	0.21	0.62	0.32
P7	0.42	0.67	0.51	P5	0.54	0.81	0.65
P8	0.46	0.79	0.58	P6	0.38	0.75	0.50

V. THREATS TO VALIDITY

Crowd workers volunteer to perform the job leading to self-selection bias. We gave workers a high rating regardless of the labels found to reduce the effects that oDesk ratings can have on the quality of the work. Additionally, our time estimates are conservative, as workers could leave the IDE open without being active. Similarly, they could have gotten outside help.

We aggregated labels manually. To reduce subjectivity, we used two independent aggregators, and two labels were considered as synonyms only when both researchers agreed.

Finally, we only investigate one task in a small project to show feasibility. Further studies are needed with larger projects and more tasks to increase the generalizability of our results.

VI. CONCLUSIONS & FUTURE WORK

We used a crowd to identify relevant skills for a task. Participants were successful in retrieving relevant labels: at 25% agreement, the crowd has a recall of 0.67 and precision of 0.76.

A majority of the labels, were project-specific, followed by labels for language constructs and programming concepts. However, programming concepts (e.g., recursion) that need understanding the program logic were not identified.

Both the ns-context and the source code are rich sources for labels. Although the source code has more false positives (relevant for the file, but not for the task). Participants spent the majority of their time scanning the (entire) source code. Therefore, we can increase the efficiency (and precision) by scoping the source code that the crowd analyzes. We plan to use fault localization [25], [26], and data mining [27] to do so.

For now, while we have shown that skill labeling is possible with a crowd-based approach, there are two open questions:

- *Where can we find a crowd of skilled workers*? We believe that skill labeling can be a good onboarding task, as experience level did not have an impact on the results. Hence, in OSS, newcomers can become familiar with the project by task labeling, before starting to make code contributions.
- *How can we make the crowd more efficient*? Automated approaches that mine the source code to create an initial set of topics or skills that the crowd can then filter may improve the results and efficiency of the labeling process.

ACKNOWLEDGMENTS

We thank Lin Huiwen, owner of *AndroidSwipeLayout* and Andrew Faulring for the Cryolite (logging) plugin.

REFERENCES

[1] M. Fazel-Zarandi and M. S. Fox, "Constructing expert profiles over time for skills management and expert finding," in *Proc. 11th Int. Conf. Knowl. Manag. Knowl. Technol. - i-KNOW '11*, 2011, p. 1.

[2] A. I. Wasserman, "Software Tools and the User Software Engineering Project," in *Softw. Dev. Tools*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1980, pp. 93–118.

[3] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in *Proceeding 28th Int. Conf. Softw. Eng. - ICSE '06*, 2006, p. 361.

[4] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage," *ACM Trans Softw Eng Methodol*, vol. 20, no. 3, pp. 1–35, Aug. 2011.

[5] G. von Krogh, S. Spaeth, and K. Lakhani, "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study," *SSRN Electron J*, vol. 32, pp. 1217–1241, Jul. 2003.

[6] N. DUCHENEAUT, "Socialization in an Open Source Software Community: A Socio-Technical Analysis," *Comput Support Coop Work*, vol. 14, no. 4, pp. 323–368, Aug. 2005.

[7] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects," in *Proc. 18th ACM Conf. Comput. Support. Coop. Work Soc. Comput. - CSCW '15*, 2015, pp. 1379–1392.

[8] I. Steinmacher, I. S. Wiese, T. Conte, M. A. Gerosa, and D. Redmiles, "The hard life of open source software project newcomers," in *Proc. 7th Int. Work. Coop. Hum. Asp. Softw. Eng. - CHASE 2014*, 2014, pp. 72–78.

[9] Y. C. Cavalcanti, I. do C. Machado, P. A. da M. S. Neto, E. S. de Almeida, and S. R. de L. Meira, "Combining rule-based and information retrieval techniques to assign software change requests," in *Proc. 29th ACM/IEEE Int. Conf. Autom. Softw. Eng. - ASE '14*, 2014, pp. 325–330.

[10] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Trans Softw Eng*, vol. 31, no. 6, pp. 429–445, Jun. 2005.

[11] S. Cooper, F. Khatib, A. Treuille, J. Barbero, J. Lee, M. Beenen, A. Leaver-Fay, D. Baker, Z. Popović, and F. Players, "Predicting protein structures with a multiplayer online game," *Nature*, vol. 466, no. 7307, pp. 756–760, Aug. 2010.

[12] J. C. Tang, M. Cebrian, N. A. Giacobe, H.-W. Kim, T. Kim, and D. B. Wickert, "Reflecting on the DARPA Red Balloon Challenge," *Commun ACM*, vol. 54, no. 4, pp. 78–85, Apr. 2011.

[13] T. D. LaToza, W. B. Towne, C. M. Adriano, and A. Van Der Hoek, "Microtask Programming: Building Software with a Crowd," in *the 27th Annual ACM Symposium on User Interface Software and Technology*, 2014, pp. 43–54.

[14] K. T. Stolee and S. Elbaum, "Exploring the use of crowdsourcing to support empirical studies in software engineering," in *Proc. 2010 ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas. - ESEM '10*, 2010, pp. 1–4.

[15] W. Dietl, S. Dietzel, M. D. Ernst, N. Mote, B. Walker, S. Cooper, T. Pavlik, and Z. Popović, "Verification games," in *Proc. 14th Work. Form. Tech. Java-like Programs - FTfJP '12*, 2012, pp. 42–49.

[16] S. Komarov, K. Reinecke, and K. Z. Gajos, "Crowdsourcing performance evaluations of user interfaces," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. - CHI '13*, 2013, p. 207.

[17] T. Sandholm, H. Ung, C. Aperjis, and B. A. Huberman, "Global budgets for local recommendations," in *Proc. fourth ACM Conf. Recomm. Syst. - RecSys '10*, 2010, p. 13.

[18] S. Park, G. Mohammadi, R. Artstein, and L.-P. Morency, "Crowdsourcing micro-level multimedia annotations," in *the ACM multimedia 2012 workshop*, 2012, pp. 29–34.

[19] A. Kulkarni, P. Gutheim, P. Narula, D. Rolnitzky, T. Parikh, and B. Hartmann, "MobileWorks: Designing for Quality in a Managed Crowdsourcing Architecture," *IEEE Internet Comput*, vol. 16, no. 5, pp. 28–35, Sep. 2012.

[20] S.-H. Kim, H. Yun, and J. S. Yi, "How to filter out random clickers in a crowdsourcing-based study?," in *Proc. 2012 BELIV Work. Beyond Time Errors - Nov. Eval. Methods Vis. - BELIV '12*, 2012, pp. 1–7.

[21] A. Sheshadri and M. Lease, "SQUARE: A Benchmark for Research on Computing Crowd Consensus," in *AAAI Conf. Hum. Comput. Crowdsourcing*, 2013, pp. 156–164.

[22] C. Macdonald and I. Ounis, "Voting for candidates," in *Proc. 15th ACM Int. Conf. Inf. Knowl. Manag. - CIKM '06*, 2006, p. 387.

[23] B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala, "DebugAdvisor," in *Proc. 7th Jt. Meet. Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng. Eur. Softw. Eng. Conf. Found. Softw. Eng. Symp. - E*, 2009, p. 373.

[24] A. Faulring and B. Myers, "Cryolite," *CMU Natural Programming*. .

[25] L. Guo, J. Lawall, and G. Muller, "Oops! where did that code snippet come from?," in *Proc. 11th Work. Conf. Min. Softw. Repos. - MSR 2014*, 2014, pp. 52–61.

[26] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization," *IEEE Trans Softw Eng*, pp. 1–1, Jan. 2016.

[27] X. Ye, R. Bunescu, and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. - FSE 2014*, 2014, pp. 689–699.