Exploring the Use of Crowdsourcing to Support Empirical Studies in Software Engineering

Kathryn T. Stolee, Sebastian Elbaum
Department of Computer Science and Engineering
University of Nebraska – Lincoln
Lincoln, NE, U.S.A.
{kstolee, elbaum}@cse.unl.edu

ABSTRACT

The power and the generality of the findings obtained through empirical studies are bounded by the number and type of participating subjects. In software engineering, obtaining a large number of adequate subjects to evaluate a technique or tool is often a major challenge. In this work we explore the use of crowdsourcing as a mechanism to address that challenge by assisting in subject recruitment. More specifically, through this work we show how we adapted a study to be performed under an infrastructure that not only makes it possible to reach a large base of users but it also provides capabilities to manage those users as the study is being conducted. We discuss the lessons we learned through this experience, which illustrate the potential and tradeoffs of crowdsourcing software engineering studies.

General Terms

Experimentation, Measurement

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics

Keywords

empirical studies, crowdsourcing

1. INTRODUCTION

Recruiting the necessary type and number of subjects is consistently a great challenge when conducting empirical studies in software engineering. As a community we understand the importance of having enough subjects of the right kind in order to obtain stronger and more general findings, yet achieving such ideal conditions is rare and usually prohibitively expensive.

As a result, we tend to compromise by performing studies on a sufficient number of subjects that may not exactly match what was intended (e.g., observing students performing a project as part of a computer science course instead of observing software engineers actually practicing), by performing studies on a limited number of subjects (e.g., evaluating a small sample of software engineers in a micro-context), by performing studies without human subjects (e.g., assessing the artifacts produced by software engineers), or by applying a combination of those strategies.

In this paper we explore the use of *crowdsourcing* as an alternative to address the challenge of recruiting the right type and number of subjects to assess a software engineering technique or tool. The basic idea behind crowdsourcing is to allow a client to leverage a global community of users with different talents and backgrounds to help perform a task (e.g., solve a problem, classify data, refine a product, gather feedback) that would not be feasible without a mass of people behind it [5]. The constant increase of people with continuous web access and the rapid maturation of the infrastructure to bring clients and producers together has started to make crowdsourcing a reality for some businesses. One of the largest and most popular infrastructure and communities available, Mechanical Turk [10], was developed and is supported by Amazon.com, and is said to have over a hundred thousand tasks posted by clients and thousands of workers signed up to complete them.

We note, however, that businesses are not the only ones starting to realize the potential for crowdsourcing. Researchers in other fields have started to use crowdsourcing to evaluate visualization designs [4], conduct surveys that ask about peoples' information seeking behaviors [2, 16], and perform natural language annotation tasks to train machine learning algorithms [14]. Furthermore, toolkits and guidelines to facilitate the crowdsourcing of user studies in the area of human computing interaction are starting to emerge [7, 8]. It seems timely, then, for our community to start exploring this trend in the context of software engineering studies where the subjects are neither in a classroom nor are they a part of a group of industry developers, but rather belong to a unknown *crowd* [12], which has the potential to offer different tradeoffs to the researcher.

The contributions of this work are: 1) a description of how we designed and adapted a real software engineering experiment to fit the popular Mechanical Turk crowdsourcing infrastructure, and 2) a discussion of the lessons we learned throughout this experience and what it means for the implementation of software engineering studies. To our knowledge, this is the first paper to provide insights on the potential of crowdsourcing to address one of the major challenges in conducting software engineering empirical studies: the recruitment of the appropriate type and number of subjects.

2. CROWDSOURCING A STUDY

In this section, we describe our study and how it was adapted to fit an infrastructure that supports crowdsourcing.

2.1 Our Study

Our objective was to assess the impact of coding practices, specifically code smells[1], on the user's preference and understandability of web mashups.[2] Mashups have become extremely popular as development environments make it possible for users to quickly get, process, and glue data with powerful APIs. For example, Yahoo! Pipes [11], one of the most popular mashup development environments, provides users with a drag and drop interface to create "pipes" by selecting and configuring predefined modules and connecting them with wires through which data flows. Over 90,000 end users have created pipes since 2007 and over 5 million pipes are executed in the Yahoo!'s servers daily [6]. We limit the scope of our study to mashups built by end user programmers, that is, programmers whose job is not software development but who do program as part of their activities [13]. We further constrain the study to end users creating mashups in the Yahoo! Pipes environment we previously mentioned.

To achieve our objective, we designed two experiments that evaluate the impact of code smells from the perspective of the end user. The first experiment aimed to determine if (RQ1) users prefer pipes with or without smells, and the second aimed to determine if (RQ2) smelly pipes are harder to understand than clean pipes. As shown by the experimental design in Figure 1, each experiment was split into a series of tasks with a random assignment of subjects to task (R). In each task, we treated one pipe, P, with a smell, X, providing coverage for a variety of common smells and pipe structures. In the tasks for the first experiment (1 – 8), the user was given two pipes side by side, one with smells and the other one without, and asked to choose the preferred pipe and explain the decision. In the tasks for the second experiment (9–10), the user was presented with either the treated or the untreated pipe and asked to determine the pipe's behavior. More details on the study design and the specifics of each experimental task are available [15].

In both experiments we estimate user aptitude by measuring education level (O_1) and qualification score (O_2) using a pretest. The pretest included questions about the users' background and eight comprehensive questions about Yahoo! Pipes. Users were required to pass the pretest prior to participation, allowing us to control for user variability. In terms of posttest measures, the first experiment evaluates user preference (O_3), the second measures correctness (O_4), and both measure the time to complete the task (O_5).

2.2 Adapting Study to Mechanical Turk

Mechanical Turk is a service that allows people to reach and compensate others to complete tasks that require human input, such as tagging images or answering survey questions. It hosts the tasks, manages payment, and makes the tasks accessible to a large and existing workforce. To assist with user recruitment and the execution of the study, we chose to take advantage of these features.

There are two roles in Mechanical Turk, a *requester* and a *worker*. The requester is the creator of a *human intelligence task*, or *HIT*, which is intended to be a small, goal-oriented task that can be accomplished in less than 60 seconds. The worker is the one who completes the HIT. Workers must discover HITs to complete by searching based on some criteria, such as title, description, keyword, reward, or expiration date. A HIT may or may not have prerequisites for the user, which are referred to as *qualifications*. Some qualifications are related to the quality of the work produced by the worker, while others can be more contextual, such as demonstrating some domain-specific knowledge. Completed tasks are returned to the requester for evaluation. If a requester is dissatisfied with submitted work, they hold the right not to pay the worker.

[1]Code smells are code characteristics whose presence may indicate deficiencies associated with higher development costs [3].

[2]A mashup is an application that manipulates and composes existing web data sources or functionality to create a new piece of data or service that can be plugged into a web page or integrated into an RSS feed aggregator.

Task	**Assignment**	**Pretest Measures**	**Object**	**Treatment**	**Posttest Measures**
1	R	O_1, O_2	P_1	X_5	O_3, O_5
2	R	O_1, O_2	P_2	X_4	O_3, O_5
3	R	O_1, O_2	P_3	X_5	O_3, O_5
4	R	O_1, O_2	P_4	X_8	O_3, O_5
5	R	O_1, O_2	P_5	X_7	O_3, O_5
6	R	O_1, O_2	P_6	X_1	O_3, O_5
7	R	O_1, O_2	P_7	X_5, X_{10}	O_3, O_5
8	R	O_1, O_2	P_8	X_2, X_7	O_3, O_5
9	R	O_1, O_2	P_9	X_6	O_4, O_5
	R	O_1, O_2	P_9		O_4, O_5
10	R	O_1, O_2	P_{10}	X_2, X_3	O_4, O_5
	R	O_1, O_2	P_{10}		O_4, O_5

Figure 1: Study Design for Experiment 1 (Tasks 1-8) and Experiment 2 (Tasks 9-10)

For each experiment, we created a HIT template, which allows for the rapid creation of multiple HITs with the same structure and the same *HIT type ID*. The type ID is used by the search interface to combine similar HITs, making them more accessible. This allowed us to present all tasks per experiment in a single entry in the search results so users could access multiple tasks without returning to the search page. The HITs for the first experiment shared a type ID, and the HITs for the second experiment shared a different type ID.

The implementation of the first experiment was straight-forward. Each experimental task mapped to exactly one HIT, and since the tasks were independent of one another (i.e., all pipes and treatments were different), they could be completed in any order. In the second experiment, we encountered some limitations imposed by Mechanical Turk. Ideally, we would have only presented a user with either the treated or the untreated pipe for each task. However, since we created a HIT template for this experiment, there were two HITs created for each task, so we could not impose a constraint that only allowed the user to perform one HIT per experimental task to control for learning effects (e.g., if a user first performs Task 9 with the untreated pipe, their familiarity with the behavior may impact their answer to Task 9 with the treated pipe). In the end, we had four HITs in the second experiment, two for each task, and allowed the user to complete all four. In the analysis we only considered the first HIT completed by a user for each task. This caused us to waste some data (and thus some cost in terms of additional rewards). An alternate design would be to create two HIT type IDs, one for each task, where each type ID has two HITs associated with it, and the user could only answer one HIT per task. However, we wanted to maintain the locality of the HITs for this experiment in the search results and included them all in the same type ID.

Our study was available for two weeks, from April 28 - May 13, 2010, and users were paid up to \$0.20 for each task. To deliver the pretest described in Section 2.1, we created a custom qualification test. Once a user submitted a qualification test, it was graded as per our specification. A passing score allowed the user to complete the HITs we created. Figure 2 shows the workflow a user had to go through to participate in our study, from creating an account to submitting completed HITs.

3. LESSONS LEARNED

There are many costs and benefits of using a service such as Mechanical Turk to conduct empirical studies of software engineering techniques and tools, some of which were mentioned informally in Section 2.2. In the end, we were able to obtain 22 qualified participants for a total cost of just under $42 that help us to determine that users consistently prefer to work with clean pipes, and that smells make it more difficult to understand how pipes work. From the perspective of this paper, however, more important than the particular outcome of the study is to share some of the lessons we learned while designing and deploying the crowdsourced study.

3.1 Recruiting Participants

Recruiting the right type and number of participants is a common challenge in software engineering studies. Our initial conjecture was that crowdsourcing could address that challenge by providing access to a large pool of candidate participants that would select to participate in the tasks we proposed as part of a study. More specifically, in Mechanical Turk, people interested in participating in a task, in our case as part of a study, find candidate tasks on their own using the search interface provide by the infrastructure. Relying on Mechanical Turk's search interface for recruitment means the researcher has less control over the users participating in the study and over variations caused, for example, by how prominently the study is displayed in the infrastructure search results.

During the first week of our study, we had each HIT completed by 6 users. Still, since we wanted to have at least 10 users per HIT, we doubled the initial monetary reward and sent emails to two internal mailing lists. At the end of two weeks, we had 50 users who opted to take the qualification test.[3] Of these users, 34 (68%) received a passing score, from which 22 (65%) followed-through with the study by completing one or more HITs. In total, we received 160 HIT responses (plus an additional 28 from the second experiment that were thrown away to control for the learning effect mentioned before), for an average of 7 HITs completed per user, and 13 users per HIT. On average, each user earned less than $2.00 for participating. We had nearly equal numbers of men and women, and most users scored at least 7 out of 8 on the qualification test. Among the 22 users, eight (36%) held degrees in computer science or related field, and the remaining 14 (64%) had only some or no computer science experience (the main target population of our study). [4]

3.2 Response Quality

One concern about crowdsourcing studies using infrastructures like Mechanical Turk is the usefulness of the collected data since user diversity, unknown experience, and people who "game" the system may impact the response quality [7]. Since Amazon anonymizes participant identities, it is also difficult to control for certain aspects of the population, such as age, gender, and education or training level, which makes other forms of screening participants particularly important. The use of a qualification test that asks such questions can mitigate this risk, but possibly at the cost of reduced participation, since the user is not rewarded to complete the qualification test and risks not passing.

[3]We note that Mason, et al. [9] indicate that additional financial incentives do increase the amount of work produced by a crowd. However, we do not know to what degree the additional reward or the email requests had an effect in the number of participants.

[4]The percentage of participants who hold computer science degrees versus those who do not, 36% to 64%, is similar to the results from a survey on 1001 Mechanical Turk participants, where 40% of the professionals declared an occupation in *Science, Engineering or IT*. and the remaining 60% declared other occupations [1].

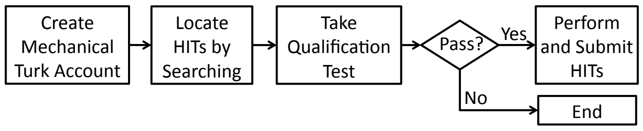

Figure 2: Applicant Workflow in Mechanical Turk

To mitigate the likelihood of participants "gaming" the system by answering the HITs haphazardly, we required a qualification test, which undoubtedly limited participation but allotted us some control over the users we were hiring, and asked users to justify their HIT responses using a textual response of at least 10 words. We expected for the textual responses to allow us to reject inadequate submissions, but that never occurred. However, this additional data helped us to better understand the users' thought processes when completing the HITs. Among the open-ended textual responses, the average number of words was over 31. We found the explanations detailed and in general demonstrating a good understanding of the questions. When the quantitative answers were homogenous for a HIT, the open-ended answers served as confirmation that the participants had an understanding of what we were asking. When the quantitative answers were dissimilar, the open-ended answers helped us understand points of confusion and why the participants differed. We could also tell when the user misunderstood the question or did not understand the task. These benefits helped us better understand the data and see that in some cases, an odd answer was the result of a misinterpretation of the question.

3.3 Support for Study Management

Mechanical Turk offers requesters three mechanisms for creating HITs: a web interface, a command line tool, and an API. Each method offers increasing levels of flexibility for the study design. For example, custom qualification tests cannot be created through the web interface, but can be created using the command line tool or API. In our study, we used the command-line tool to create HITs and evaluate qualification tests, but used the web interface to approve work completed and access the results. The command line tools require knowledge of XML, web services, and shell scripting, and a credit card is needed to front-load the requester account with funds to conduct the study. The HIT creation can be tested in the developer sandbox prior to deployment on the live server.

Beyond support for study implementation, we found that using established services to crowdsource empirical studies offers several other benefits to the researcher. By providing a framework on which the study can be conducted, several aspects of the study, including recruitment, ensuring privacy, distributing payment, and collecting results, were more easily managed. Further, in Mechanical Turk, all identities are anonymized so that the collected data cannot be associated with the users identity, which preserves the privacy requirements we had set. Results are collected by the infrastructure and can be easily downloaded in csv format by the researchers. Removing these concerns saved us time in the implementation and execution of a study.

However, crowdsourcing a study through an established service like Mechanical Turk does require overcoming a learning curve.

Even if the researcher has access to programmers to assist in the implementation of the study under a desired infrastructure, some understanding of its capabilities and constraints is required to determine how to best adapt the study to it, and how that may impact the design and ultimately the study findings. That said, such infrastructure requires and hence forces a detailed planning of the study execution, which in our experience later reduced the need to be present as the study was been conducted. Clearly, some overseeing of the evolution of the study to detect potential anomalies and abuses is required, but this freedom enables the execution of longer studies on which a larger number of users can participate.

3.4 Adjustments in Design and Operation

As with any experimental setting, operating within an infrastructure like Mechanical Turk poses some constraints that must be clearly understood and mitigated. For example, one of such constraints is for the study to be broken into a series of tasks. Even then, our study uses tasks that are much more complex and time consuming (up to 10 times more) than those recommended by the Mechanical Turk guidelines. Now, since the order in which these tasks are completed cannot be easily enforced, the tasks must be independent of one another and the researcher must evaluate if a randomized assignment of subjects to tasks is appropriate for their study. Along those same lines, if a user study contains multiple tasks, it cannot be guaranteed that each user will perform each task. Due to these limitations, the researcher must consider for learning effects. For example, in our preference experiments, the sample pipes given in each task had to be different so a user was forced to learn about a new pipe for each HIT. Other constraints are the ability to capture the context under which the participants completed the tasks, which can be a powerful factor in the results, and the ability to collect certain metrics for the tasks being performed by participants, most specifically the time to completion. While time to completion is reported by Mechanical Turk in the result sets, it may not accurately measure the task time in part because we do not get a sense for how much time the user spent on the task and how much time the task was simply open in the browser.

We note, however, that many of the design limitations can be mitigated by using Mechanical Turk as a front end to recruit users and manage payment, while implementing the actual study at a third-party site and including a pointer to that site within a Mechanical Turk HIT. Once the user finishes the activity at the site, they could collect a token and provide it to Mechanical Turk to complete a HIT. Clearly, this involves additional effort since some of the support services of the infrastructure are not used, but the access to the large pool of users to crowdsource the study still remains.

4. CONCLUSION

In this work we set out to convey our experiences in crowdsourcing a software engineering study. We found that the benefits of an infrastructure like Mechanical Turk to access and manage a large pool of study participants are enticing. We have also identified several issues that must be taken into consideration for the implementation of effective crowdsourced studies. These issues include the additional controls to ensure the right participation of qualified subjects and the quality of the responses, and the extra effort on the part of the researcher to learn a new infrastructure, to assess its appropriateness for a study requirements especially in terms of control, and to tailor the study to fit the infrastructure constraints and capabilities. Based on this preliminary experience, we believe that crowdsourcing provides an alternative approach with unique tradeoffs for conducting empirical studies in software engineering.

5. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation through a Graduate Research Fellowship to Kathryn Stolee, and NSF-0915526.

6. REFERENCES

[1] Julie S. Downs, Mandy B. Holbrook, Steve Sheng, and Lorrie Faith Cranor. Are your participants gaming the system?: screening Mechanical Turk workers. In *Proceedings of the 28th international conference on Human factors in computing systems*, 2010.

[2] Brynn M. Evans and Ed H. Chi. Towards a model of understanding social search. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, 2008.

[3] Martin Fowler and Kent Beck. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[4] Jeffrey Heer and Michael Bostock. Crowdsourcing graphical perception: using Mechanical Turk to assess visualization design. In *Proceedings of the 28th international conference on Human factors in computing systems*, 2010.

[5] Jeff Howe. The rise of crowdsourcing. *Wired Magazine*, 14(06):17–23, 2006.

[6] M. Cameron Jones and Elizabeth F. Churchill. Conversations in Developer Communities: A Preliminary Analysis of the Yahoo! Pipes Community. In *Proceedings of the Fourth International Conference on Communities and Technologies*, 2009.

[7] Aniket Kittur, Ed H. Chi, and Bongwon Suh. Crowdsourcing user studies with Mechanical Turk. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, 2008.

[8] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkit: tools for iterative tasks on Mechanical Turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, 2009.

[9] W. Mason and D.J. Watts. Financial Incentives and the Performance of Crowds. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 77–85. ACM, 2009.

[10] Amazon Mechanical Turk. https://www.mturk.com/mturk/welcome, June 2010.

[11] Yahoo! Pipes. http://pipes.yahoo.com/, July 2009.

[12] Joel Ross, Lilly Irani, M. Six Silberman, Andrew Zaldivar, and Bill Tomlinson. Who are the crowdworkers?: shifting demographics in Mechanical Turk. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, 2010.

[13] Christopher Scaffidi, Mary Shaw, and Brad Myers. Estimating the numbers of end users and end user programmers. In *Symposium on Visual Languages and Human Centric Computing*, 2005.

[14] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2008.

[15] Kathryn T. Stolee. Analysis and transformation of pipe-like web mashups for end user programmers. Master's Thesis, University of Nebraska–Lincoln, June 2010.

[16] Survey Vault. http://www.surveyvault.co.uk/, July 2010.