Exploring the Benefits of Using Redundant Responses in Crowdsourced Evaluations

Kathryn T. Stolee, James Saylor, and Trevor Lund
Department of Computer Science
Iowa State University
{kstolee, jsaylor, tlund}@iastate.edu

***Abstract*—Crowdsourcing can be an efficient and cost-effective way to evaluate software engineering research, particularly when the evaluation can be broken down into small, independent tasks. In prior work, we crowdsourced evaluations for a refactoring technique for web mashups and for a source code search engine, both using Amazon's Mechanical Turk. In the refactoring study, preference information was gathered when comparing a refactored with an unrefactored pipe, in addition to a free-text justification. In the code search study, information was gathered about whether a code snippet was relevant to a programming task and why. In both studies, we used redundant metrics and gathered quantitative and qualitative data in an effort to control response quality. Our prior work only analyzed the quantitative results.**

In this work, we explore the value of using such redundant metrics in crowdsourced evaluations. We code the free-text responses to unveil common themes among the responses and then compare those themes with the quantitative results. Our findings indicate high similarity between the quantitative and free-text responses, that the quantitative results are sometimes more positive than the free-text response, and that some of the qualitative responses point to potential inadequacies with the quantitative questions from the studies.

I. Introduction

Research in software engineering often requires evaluation using human subjects. The cost of designing and running these studies is high, leading researchers to explore more cost-effective strategies. From our experience, crowdsourcing evaluations has increased our efficiency and decreased our costs to run evaluations, allowing us to evaluate research ideas with dozens of participants for a fraction of the monetary costs of in-person studies.

Even with these benefits, crowdsourcing evaluations must be done carefully, are not appropriate for all evaluations, and have tradeoffs. These tradeoffs primarily manifest in losses of control, specifically over the target population, the environment in which the participants perform tasks, duration the study will last, and the types of information that can be gathered. In an effort to obtain better quality results, we have used redundant metrics in the form of quantitative and free-text responses.

We have used this approach for two different projects, one with the goal of determining whether users of Yahoo! Pipes prefer refactored or smelly pipes [1], [2], and the other determining whether or not search results are relevant to particular programming tasks [3]. For the former, our quantitative results have shown users generally prefer pipes without smells. For the latter, we found that given a smart ranking algorithm, our search approach [4] returned more relevant results than a search approach powered by Google. These general results were based on quantitative analyses of the user responses, but we also collected free-text responses explaining those results. These responses were intended to prevent participants from "gaming" the system and performing the tasks haphazardly. In this work, we explore the free-text results and determine if indeed they served that purpose. Our results indicate that the textual responses generally support the quantitative results and provide supplementary information that enriches the findings.

The contributions of this work are:

- Identification of common themes from free-text responses in two crowdsourced evaluations on source code relevance and refactoring preferences
- Comparison of qualitative and quantitative redundant metrics from two crowdsourced evaluations

II. Methodology

Both studies were crowdsourced using Amazon's Mechanical Turk [5], a service advertised as a "marketplace for work that requires human intelligence." 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. The worker completes the HITs for payment.

In our past studies, we had included free-text responses in our HITs with the idea of deterring people from answering the HITs haphazardly. Up until now, we have only reported on the quantitative results. In this work we are interested in learning *how valuable is it to include free-text responses in crowdsourced software engineering evaluations*.

We analyze the data sets from the two studies in two ways, first by extracting themes from the free-text responses, and second by looking for congruence between the quantitative and qualitative responses. For both studies, we coded the free form responses to identify common themes. Two passes were completed over each data set. In the first, one author identified a set of themes. The themes were discussed among the authors to resolve any questions. In the next pass, the same author assigned the themes to the responses. This was done via manual analysis.

The results explore the themes extracted and compare those themes to the quantitative results from our prior work. We discuss each study separately, starting with the refactoring study in Section 3 and the code search study in Section 4.

III. Analyzing Refactoring Preferences

From a research perspective, code smells are indicative of deficiencies in source code and refactorings are intended to make the code better with respect to some goal, such as understandability or maintainability. For web mashups, we have explored whether or not refactored pipes are preferable to users [1], [2] by crowdsourcing the evaluation using Amazon's Mechanical Turk.

A. Original Study

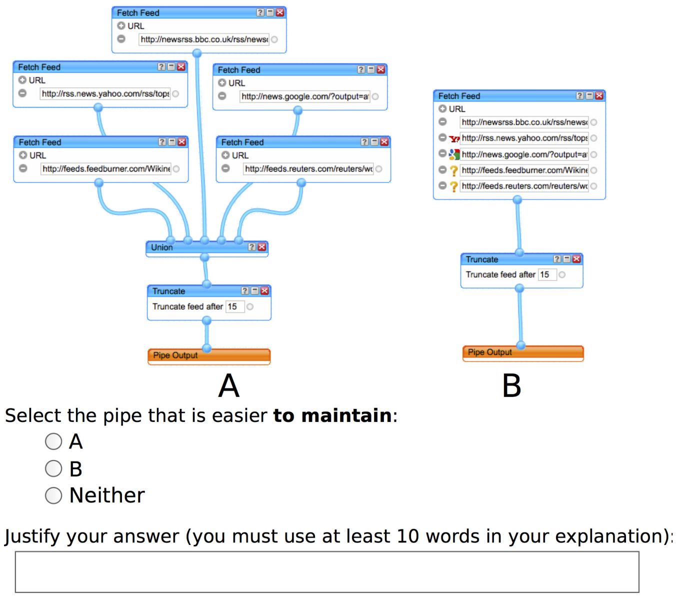
Yahoo! Pipes [6] is a visual data flow language mashup language that allows programmers to combine, filter, sort and annotate RSS feeds and other data sources. Two example pipes are shown in Figure 1. The pipe labeled A has five data source modules that contain URLs pointing to RSS feeds. These five modules feed to a union module that concatenates the data. This feeds to a sort module and then to the output. The pipe labeled B has five data sources in a single module that feeds to a sort before the output. These pipes are equivalent in that the outputs are the same. In prior work, we defined families of code smells and refactorings to transform pipes into their semantic equivalents [1], [2]. For example, the *collapse duplicate paths* refactoring would transform pipe A to pipe B. This refactoring introduces an abstraction (i.e., representing five modules with just one) and also decreases the size of the pipe (i.e., from eight modules to three). The overall goal of this study was to determine if end-user programmers preferred one pipe over the other, and why.

1) Implementation: We designed tasks in which a participant is presented with a pipe that contains a code smell and one that does not, and are asked which is preferred with respect to a goal (e.g., to understand or to maintain). Next, they have to justify their decision using a free-form text box. An example is shown in Figure 1, which asks which pipe is easier to maintain, A, B, or neither. Each task we defined for this study was implemented as a HIT on Mechanical Turk, and participants were paid $0.25 per HIT completed. Multiple HITs could be completed per participant; in total there were 17 different HITs.

Each participant was required to take and pass a qualification test. This test is used to control for participant quality by asking questions about Yahoo! Pipes, collect demographic information about the participants (e.g., education level), and obtain Institutional Review Board[1] consent, per our institution's policy. A passing score allowed the user to complete any of the HITs in this study. Retakes on the qualification exam were not permitted in the event of a failing score.

2) Participants: The survey and qualification test were completed by 258 subjects and 135 (52%) received a passing score of 50% or more. A total of 61 subjects participated in the study.

[1]The Institutional Review Board, or IRB, is an organization that reviews and authorizes study protocol for experiments involving human subjects to protect the volunteer subjects.


Fetch Feed
URL
http://newsrss.bbc.co.uk/rss/newso
http://rss.news.yahoo.com/rss/tops
http://news.google.com/?output=a
http://feeds.feedburner.com/Wikin
http://feeds.reuters.com/reuters/w
Union
Truncate
Truncate feed after 15
Pipe Output
A
B

Select the pipe that is easier **to maintain**:
- ○ A
- ○ B
- ○ Neither

Justify your answer (you must use at least 10 words in your explanation):

Fig. 1. Illustration of Refactoring Preference Task

3) Data: In total, we analyzed 333 qualitative responses from 17 tasks. On average, each response had 29.58 words with $\sigma = 16.57$. Of the 333 responses, 139 from seven HITs were related to understandability and 194 from 10 HITs were related to maintainability. Note that compared to previous work that focused on the quantitative analysis of these data [2], three HITs did not include justifications and were thrown out for this analysis.

B. Evaluation

1) Theme Extraction: We found 10 themes that represent 96% of the textual responses, and include a catch-all theme, *Other* (theme `k`), to capture the rest. Only one theme was assigned per response. The themes and their lower-case letter abbreviations are listed in Table I. As an example of the coding process, the response, *"abstraction of values through string builder module and providing them through wire makes the maintenance easy."*, was assigned to theme `a` in which *abstraction aids maintainability*. As another example, the response, *"B can be updated easily as it contains string builder separately for each modules"* was also assigned to the theme `a` in which *abstraction aids maintainability* since the *string builder* module is an added abstraction. The response, *"You need the same number of operations to update any of the pipes."*, was assigned to theme `i` where the *pipes are identical*.

2) Comparing Quantitative and Qualitative: Participants specified which pipe was preferable and then explained why. For this part of the analysis, we looked at the common themes and compared those to the quantitative results. We compare to the results published in our prior work [1], [2].

C. Results

1) Themes: Most of the free-text responses could be coded, but four of the 333 responses were nonsensical and grouped

TABLE I
THEMES FOR REFACTORING STUDY RESPONSES

a	Abstraction aids maintainability
b	Abstraction makes maintainability harder
c	Updating is easier with fewer modules/items to update
d	Errors and error-prone pipes are harder to understand/maintain
e	Familiar elements are better
f	Fewer modules are easier to understand
g	More modules are easier to understand
h	Explicit, hard-coded values are easier to understand
i	Pipes are identical
j	Naming conventions
k	Other

TABLE II
CODING RESULTS FOR REFACTORING STUDY. KEY FOR THEMES IN TABLE I. DOMINANT THEMES ARE BOLDED FOR EACH TASK.

Task	a	b	c	d	e	f	g	h	i	j	k
1	3	3	**14**	0	0	0	0	0	0	0	0
2	**16**	0	11	2	0	0	0	0	0	0	1
3	8	1	**10**	0	0	1	0	0	5	0	5
4	**12**	0	2	1	0	0	0	2	3	0	0
5	**15**	0	2	0	0	1	1	0	1	0	0
6	1	0	2	2	1	0	0	0	**9**	0	0
7	0	0	0	0	**9**	2	2	0	1	0	1
8	0	0	0	1	0	**10**	0	0	7	0	1
9	0	0	0	**15**	0	0	0	0	4	0	1
10	1	0	1	0	**13**	0	0	0	2	1	2
11	0	0	0	0	0	**9**	2	1	5	0	0
12	0	0	0	0	2	**13**	1	0	0	0	1
13	0	0	0	0	**5**	3	**5**	0	1	0	1
14	0	0	0	0	0	8	0	**12**	0	0	0
15	0	0	0	7	0	2	0	0	**12**	0	0
16	0	0	0	0	**11**	4	0	0	1	3	1
17	0	0	0	0	**9**	6	0	0	0	0	0
Sum	56	4	42	27	50	60	10	15	51	4	14
%	17	1	13	8	15	18	3	5	15	1	4

into the *other* category (theme k). The 17 tasks are each shown in Table II, along with the number of responses that pertained to each theme, identified in Table I. The first 10 tasks asked participants which pipe was easier to maintain whereas tasks 11-17 asked which pipe was easier to understand.

Using task 1 as an example (shown in Figure 1), we see that 14 responses correspond to theme c, where *updating is easier with fewer modules/items to update*. As an example, one response states, *"It is easier and quicker to add and delete feeds from pipe B's single Fetch Feed module than it is to deal with multiple modules."*

For each HIT, there tend to be a few dominant themes that represent most of the results. For example, 14 / 20 responses for task 1 map to theme c, 27 / 29 responses for task 2 map to themes a and c (which are highly similar, except theme a recognizes abstraction as the reason for easier maintenance), and 13 / 17 of the responses for task 12 map to theme f.

A majority of the responses for tasks 1-5, which deal with maintenance, indicate that *abstraction aids maintainability* (theme a) and *updating is easier with fewer modules/items to update* (theme c). This seems to favor smaller, more abstract pipes. For tasks 7-10, also dealing with maintenance, the majority themes are *errors ... are harder to maintain* (theme d), *familiar elements are better* (theme e), and *fewer modules are easier to understand* (theme f). These responses imply that understandability aids maintainability, and that less error-prone pipes with familiar elements are better. The remaining maintenance task is 6, in which participants found that the *pipes are identical* (theme i).

The high prevalence of theme f in the understandability tasks 11 and 12 indicates that smaller pipes are easier to understand. For task 13, on the other hand, there was disagreement. Five responses mapped to theme g where *more modules are easier to understand* whereas three responses mapped to theme f that *fewer modules are easier to understand*. In general, though, there seems to be high agreement among the responses when aggregated across participants.

Some themes were expected, such as *more modules are easier to understand*, since more modules typically indicates less abstraction, and we had hypothesized that abstraction may sometimes impede understandability. Other themes, such as *familiar elements are better* (theme e), was less expected. For example, with task 16, the pipes were similar except the naming on the looping module. Theme e dominated the responses for this task, saying the term "loop" was more familiar, and some participants specifically pointed to the better naming conventions (theme j). Based on this result, it would seem that the more familiar the constructs, the more understandable the pipe. In our refactoring work, we did not define a code smell for the presence of unfamiliar elements [1], [2]. Given this theme that emerged from the responses, the follow-up would be to define a new code smell based on unfamiliar elements, and we leave this for future work.

Another observation is that none of the understanding questions were answered with respect to maintenance considerations, but often understandability played into the decision on which pipe was easier to maintain. For example, *"[larger structures] are very easy to understand and one can make any update easily in the future"* was a justification given for task 5, mapping to theme g since the larger structures had more modules.

2) Quantitative vs. Qualitative: In combining the qualitative and quantitative responses, we find that occasionally the quantitative response contradicts the textual response, but generally the responses were in agreement.

In our prior work [2], we found programmers preferred the refactored pipe to the unrefactored pipe for 72% of the tasks. Exceptions to participants preferring the refactored pipe include task 6, which presents two identical pipes in structure except some of the websites accessed return `404 not found` errors. For this task, neither pipe was preferred[2], and the dominant theme is that the *pipes are identical* (theme i). Another exception is task 13, which presents a pipe with five modules and a refactored pipe where three modules are combined into one. Here, the unrefactored pipe was preferred[3] and the themes identify new code smells to enhance our prior refactoring work. Task 14 had a refactored pipe with an added module to hold a

[2]Task 4 in prior work [2]
[3]Task 7 in prior work [2]

common value to several modules. In the quantitative analysis, participants also showed a preference toward the unrefactored pipe[4], and the themes confirm that. The *explicit, hard-coded values* (theme h) were easier to understand, yet these were abstracted in the refactored pipe.

As for disagreement between the quantitative and qualitative responses, there was only one instance of obvious disagreement. For task 1, one participant selected the quantitative response, *either*, but the free-text response stated, *"Both are very basic, so although B has less windows, A is very easy to maintain"* (mapping to theme c). As this was the only instance, the impact on our prior results is minor. Beyond that, there were some cases of *potential* disagreement. For example, a participant stated, *"I would assume they are both just as easy to update in the future. Pipe A will probably just receive more duplicate records . . . which would result in more output if not filtered."* The justification somewhat indicates that pipe A will be harder to update in the future, but passes that effort off as being the same as pipe B and answered *either*.

In the end, we found that our prior work on refactoring did not capture all the code smells that matter to users. The quantitative responses showed that refactored pipes are generally preferred [1], [2], but this analysis shows that we missed some opportunities to define additional smells and refactorings that matter to the users.

IV. Analyzing the Relevance of Code Search Results

Programmers frequently search for source code using general purpose search engines. The search query is composed in a textual format, which may not be ideal for programmers searching for example code to reuse. We developed a search approach called Satsy that allows programmers to search for source code by input/output examples, similar to test cases [3], [4], [7]. We evaluated how relevant the search results are to particular Java programming tasks by crowdsourcing the evaluation using Amazon's Mechanical Turk.

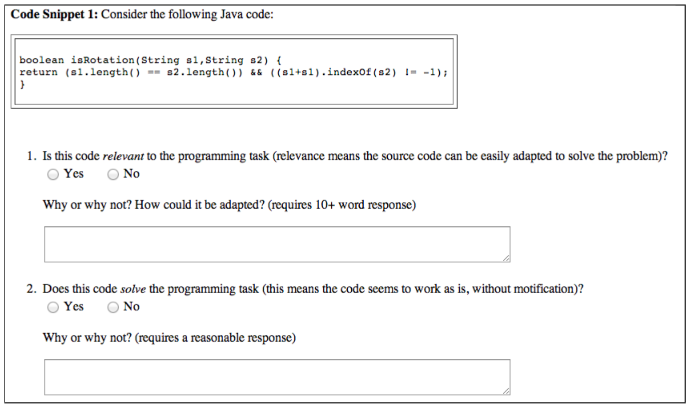
A. Original Study

The goal of this experiment was to compare the relevance of search results from Google, a code-specific search engine Merobase, and Satsy. To do this, we gathered eight Java programming tasks from stackoverflow.com and had programmers generate queries in the formats required for each of the search approaches. We then executed the queries and obtained a source code snippet (i.e., method or block levels of abstraction) from each of the top ten results. These were the code snippets evaluated by participants on Mechanical Turk for relevance.

A list of the eight programming tasks is shown in Table III. All tasks deal with string and integer manipulation.

1) Implementation: For each programming task, we presented the participant with three code snippets where one snippet came from each search approach. For each snippet, we asked whether the code was relevant to the programming task

[4]Task 17 in prior work [2]


Code Snippet 1: Consider the following Java code:

```
boolean isRotation(String s1,String s2) {
return (s1.length() == s2.length()) && ((s1+s1).indexOf(s2) != -1);
}
```

1. Is this code *relevant* to the programming task (relevance means the source code can be easily adapted to solve the problem)?
   Yes No

   Why or why not? How could it be adapted? (requires 10+ word response)

2. Does this code *solve* the programming task (this means the code seems to work as is, without motification)?
   Yes No

   Why or why not? (requires a reasonable response)

Fig. 2. Illustration of Source Code Relevance Task

using a yes/no response, and why. We also asked whether the code solved the programming task, and why.

Each HIT on Mechanical Turk contained all eight programming tasks. Thus, for a single HIT, a participant had to evaluate 24 code snippets and were paid \$3.25 for completing the HIT. Only one HIT could be completed per participant.

A participants' workflow is similar to the refactoring study, described in Section III-A1, where participants must complete a qualification test. In this case, the test had four questions about the Java programming language. Retakes were not permitted.

An example of a single code snippet and the quantitative and qualitative questions is shown in Figure 2; the relevant programming task is task 8 in Table III.

2) Participation: We had a total of 30 HITs and 30 participants. We did not keep track of how many participants failed the qualification test for this study.

3) Data: For the quantitative results [3], we analyzed the data from 720 snippets with respect to whether the code is relevant and/or solves a programming task. For this qualitative analysis, we analyze the relevance and solves responses for 244 snippets, totaling 488 free-text response evaluations. This covers approximately 33% of the total responses for the study, sampled nearly uniformly from the tasks and search engines. We break this data down by programming task and search approach in Table III. For each combination of programming task and search approach, we evaluated at least 10 snippets.

Among the free-text responses, the average relevance response had 21.9 words with $\sigma = 13.53$. The average solves response had 15.71 words with $\sigma = 10.85$.

B. Evaluation

1) Theme Extraction: For each snippet evaluated, two textual responses were given, one explaining whether the snippet is *relevant* to the programming task, and another explaining whether the snippet *solves* the programming task (Figure 2). We aggregated the themes assigned for the two responses since there was high repetition.

In all, we found 35 themes that represent the responses, and each response can have more than one theme. The free-text response often contained many parts that were hard to

TABLE III
CODE SEARCH RESPONSES ANALYZED PER PROGRAMMING TASK

	Programming Task	Google	Merobase	Satsy	Total
1	Check if one string contains another string, ignoring case (case-insensitive)	10	10	10	30
2	Given a string, capitalize the first letter.	10	10	10	30
3	Determine if an integer is positive.	10	10	10	30
4	Given the String representation of a file name, trim off the extension	10	10	10	30
5	Given a string, trim off the last character.	10	10	10	30
6	Get the character representation of a String.	10	11	10	31
7	Determine if a character is numeric.	10	10	10	30
8	Check if one string is a rotation of another string (a rotation is when the first part of a string is spliced off and tacked onto the end	11	11	11	33
	Sum	81	82	81	244

capture with a single theme. The themes and their upper-case or symbol abbreviations are shown in Table IV. As an example of the coding process, consider the following response for programming task 2: *"this is not relevant and this logic doesn't work properly."*. This response was assigned theme `H`, since it indicates the behavior (i.e., logic) is not relevant/useful to the task. As a more positive example, here's a response from programming task 7: *"this is relevant but need some more modifications."*. This was assigned themes `G` and `E`, since it is *exhibits useful behavior*, but also needs modification and is thus *nearly complete*.

We observe that there is a large difference in the number of themes extracted for the results from the two studies. We offer the following insights on this. First, there was much higher uniformity in the responses for the refactoring study than there were for the code search responses. In the refactoring study, programmers were comparing two pipes for preference, so the responses could simply point out the differences. Second, the Pipes language is visual and less expressive than Java. For the code search study, the participant had to explain why Java code does or does not work, which can be approached from many angles. For example, some explanations point to what the source code does and why it is wrong (theme `#`), whereas others point to what the code does not do (theme `$`).

2) Comparing Quantitative and Qualitative: Participants could only answer yes or no; there was no in-between response. An expectation is that an answer of *yes* for relevance would mean the textual response also indicates *yes*. In this analysis, we looked for general agreement compared to the quantitative responses as presented in prior work [3].

TABLE IV
THEMES FOR CODE SEARCH STUDY RESPONSES

A	Works for all values as-is
B	Works for all values at some point
C	Works for 1+ values as-is
D	Works for 1+ values at some point
E	Nearly complete
F	far from complete
G	Exhibits useful behavior
H	Exhibits behavior unuseful to this task
I	Exhibits behavior unlikely ever to be useful
J	Exhibits behavior that would work, but is not optimal
K	No notable behavior
L	Makes high-level recommendation
M	Makes functional recommendation to a specific line of code
N	Makes recommendation re: naming conventions
O	Makes recommendation re: organization of code
P	Makes recommendation re: algorithm change
Q	Incorrect output type
R	Incorrect output medium (e.g., print vs. return)
S	Constant output
T	Positive note on output
U	Incorrect input type
V	Note on input medium (e.g., global variable)
W	Input is fixed (at least one)
X	Other negative input note
Y	Positive note on input
Z	Not Java
!	Does not compile/runtime error
@	Missing a method
#	Describes program's capabilities
$	Describes program's deficiencies
%	Describes program's behavior
^	Program cannot be salvaged
+	Maybe (hopeful)
-	Maybe (doubtful)
)	Maybe (confusing)

C. Results

1) Themes: The themes are listed in Table IV and Table V presents the frequency of occurrence of the most common 17 themes, that is, the themes that appear in at least 10% of the responses. These are broken down by search algorithm. For example, theme `E`, which states that the snippet is *nearly complete*, appears in 51 responses, representing 21% of the snippets. Breaking this down by search approach, 19 came from Google results, 17 from Merobase, and 15 from Satsy. In Table VI, the theme frequencies are broken down by programming task. For example, theme `Z` appears in 10 of the responses for programming task 1.

The most common theme was `H`, indicating that the snippet *exhibits behavior unuseful to the task*. This is most commonly found among the Merobase snippets, followed by the Satsy snippets and then those from Google. This is in contrast to the next most common theme, `G`, which states the program *exhibits useful behavior*. Many of the responses also describe the program's behavior as a component of the explanation (theme `%`). Surprisingly, approximately 10% of the responses make a recommendation about naming conventions (theme `N`).

2) Quantitative vs. Qualitative: The search approach with the highest relevance score overall was Google, with 67.5%

of the results being marked as relevant [3, p.139]. This is consistent with our results here, where Table V shows that 45 (56%) of the responses matched theme G, where the behavior is useful. Google was also the mostly likely to provide a result that works as-is (theme A), where 25 responses had that theme compared to five from Merobase and 13 from Satsy.

Theme H indicates that the behavior of the snippet is not useful for the task and 131 (55%) of all snippets had a response that indicated this. Based on the quantitative analysis, 52% of all the snippets were found to be relevant [3, p.139], and thus 48% were irrelevant/unuseful. Theme G, that the snippet *exhibits useful behavior*, is present in 39% of the responses. The take-away message here is that the quantitative responses were slightly more positive than the qualitative.

Table VI presents the results broken down by programming task. Two-thirds of the responses we evaluated for tasks 2 and 3 indicated theme H, where the behavior is unuseful, yet the overall relevance of the snippets as found in prior work was 51% and 44%, respectively. Again, the quantitative results are more positive than the qualitative.

Of the responses analyzed, programming task 4 was the most likely to have results that work as-is (theme A), yet its quantitative relevance score was only third highest among all the programming tasks [3, p.139]

Among the quantitative responses, task 8 had the lowest quantitative score, where only 40% of the results were found to be relevant [3, p.139]. Considering the qualitative responses, 9 responses (27% of those we analyzed) described the deficiencies of the code (theme $), which could be considered in future work when tuning our search approach to better meet user needs. The quantitative responses for tasks 5 and 6 were the most relevant overall, with over 60% relevance [3, p.139]. Considering the free-text responses, rather than focusing on the code's deficiencies, over 25% of the responses we analyzed discussed the code's capabilities (theme #).

We also found is that the quantitative responses were perhaps too narrow, as indicated by the presence of theme C where the code will only work for a small set of potential inputs. For example, for the task, *Get the character representation of a String.*, one participant indicated the code snippet,

```
Character toCharacter(String self) {
        return self.charAt(0);
    }
```

would solve the problem, *"...only with strings of length 1."*, and answered *yes* that it would solve the problem.

In the end, we found that the yes/no response format for these tasks may have been too narrow. Future work should incorporate an in-between response or likert-scale response to capture the degree of relevance between a snippet and a task.

V. Discussion

Overall, the results indicate the qualitative and quantitative results provide similar results, though for the code search study in which the quantitative responses were yes/no with no in-between, the quantitative responses were more positive than the free-text responses. There were only a few instances where there was disagreement between the responses. This could mean one of two things: either the inclusion of the free-text responses in the HITs serves the intended purpose of preventing people from answering randomly, or the free-text responses are largely unnecessary. Replications of these studies without the free-text responses are needed to compare results.

As with most empirical studies, there appears to be a tradeoff between the expressiveness of the results and the cost of the study. When the quantitative results are unexpected, the free-text responses provide insights that would not otherwise be gathered. On the other hand, the presence of the free-text responses may deter a larger group of participants from completing the HITs, and are more difficult to analyze.

At the very least, qualitative responses are very useful for pilot studies. It can show that perhaps the quantitative responses are too narrow, or give insights to why the results were different than expected, allowing the study designer to refine the tasks and study structure as needed.

There still remain some open questions for how to handle certain scenarios we encountered:

1) When there is disagreement between the quantitative and free-text response, which should be used in an analysis? Should the response be thrown out?

TABLE V
Popularity of Themes in Search Responses, by Search Approach

Theme	Google	Merobase	Satsy	Total	
H	30	52	49	131	54%
G	45	21	30	96	39%
%	21	20	25	66	27%
E	19	17	15	51	21%
Z	22	14	13	49	20%
#	20	14	15	49	20%
K	16	16	14	46	19%
$	11	19	16	46	19%
A	25	5	13	43	18%
C	11	11	10	32	13%
D	11	7	9	27	11%
F	8	7	12	27	11%
N	5	11	8	24	10%

TABLE VI
Popularity of Themes in Search Responses, by Programming Task

	Programming Task								
	1	2	3	4	5	6	7	8	Total
H	13	20	20	15	16	14	15	18	131
G	13	10	10	14	14	17	10	8	96
%	8	9	6	8	8	9	10	8	66
E	7	5	9	6	8	10	3	3	51
Z	10	7	7	8	4	6	5	2	49
#	6	4	6	6	8	8	4	7	49
K	10	5	5	4	6	7	6	3	46
$	4	6	5	5	3	6	8	9	46
A	5	5	3	10	4	7	4	5	43
C	4	2	6	5	3	6	3	3	32
D	2	2	5	5	3	6	2	2	27
F	2	4	3	2	3	1	7	5	27
N	2	2	4	2	4	7	2	1	24

2) What is the impact of including the free-text responses on participants likelihood of completing HITs?
3) Does the free-text response actually control participant quality, or is it redundant?

VI. Threats to Validity

The threats to validity of this work are inherited from the original studies [1]–[3]. In addition:

A. Internal

For the code search study, we only analyzed 33% of the qualitative responses. While sampling was uniform and no bias was intentionally introduced, some of the differences between the quantitative results from prior work and qualitative results in this work may be attributed to the sample.

In the coding process, we may have introduced bias since we knew the results from the original studies. To mitigate this, the authors who coded the free-text responses were not involved in designing or analyzing the original studies.

B. External

The observations we made may not translate to other crowdsourced studies. To mitigate this, we considered two different crowdsourced studies with different research objectives to reach more general conclusions.

VII. Related Work

Researchers have sought to use crowdsourcing to complete various software development tasks, such as GUI testing [8], software verification [9], or programming tasks [10]. Our approach is different, in that we use crowdsourcing as a way to recruit participants and evaluate our research. In prior work, we reported on our initial impressions and lessons learned using crowdsourcing for one software engineering study [11].

In this work, we compare the internal consistency of quantitative and qualitative responses in two crowdsourced studies. Other work has compared the quality of a crowdsourcing study to the same study performed in a controlled lab setting [12]. The results from crowdsourcing were not as good as a lab study, primarily because the ability to ask follow-up or clarification questions was stunted.

The goal of including redundant metrics in our studies was to control participant quality. Kim et al. evaluated the impact of payment schemas on random clickers in crowdsourced studies [13]. They found that higher payments discourage random clickers, but the increase in quality may not justify the additional costs. This could be another approach to controlling user quality beyond using free-text responses.

VIII. Conclusion

Crowdsourcing software engineering evaluations can be an economical way to recruit participants and evaluate research that needs human opinions. In our prior work, we crowdsourced evaluations for two goals: evaluating the impact of refatorings on developers' preferences in Yahoo! Pipes, and evaluating the relevance of source code snippets to various programing tasks. In both studies, we used redundant metrics to control participant quality, requiring quantitative and free-text responses.

In this work, we explore the free-text responses. We found high consistency between the quantitative and textual responses, indicating that either the free-text responses served the intended purpose, or were unnecessary. While the free-text responses may deter participants and are more difficult to analyze, the responses can inform refinements to an existing study or research. Future work is needed to replicate our studies and remove the free-text response to observe the impact on the responses.

Acknowledgements

Special thanks to Sebastian Elbaum for his guidance in designing the original studies. This work is supported in part by NSF SHF-1218265, NSF SHF-EAGER-1446932, and the Harpole-Pentair endowment at Iowa State University.

References

[1] K. T. Stolee and S. Elbaum, "Refactoring pipe-like mashups for end-user programmers," in *International Conference on Software Engineering*, 2011.

[2] ——, "Identification, impact, and refactoring of smells in pipe-like web mashups," *IEEE Trans. Softw. Eng.*, vol. 39, no. 12, pp. 1654–1679, Dec. 2013. [Online]. Available: http://dx.doi.org/10.1109/TSE.2013.42

[3] K. T. Stolee, "Solving the Search for Source Code," PhD Thesis, University of Nebraska–Lincoln, August 2013.

[4] K. T. Stolee, S. Elbaum, and D. Dobos, "Solving the search for source code," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 3, pp. 26:1–26:45, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2581377

[5] "Amazon Mechanical Turk," https://www.mturk.com/mturk/welcome, June 2010. [Online]. Available: https://www.mturk.com/mturk/welcome

[6] "Yahoo! Pipes," http://pipes.yahoo.com/, June 2012.

[7] K. T. Stolee and S. Elbaum, "Toward semantic search via smt solver," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 25:1–25:4.

[8] E. Dolstra, R. Vliegendhart, and J. Pouwelse, "Crowdsourcing gui tests," in *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, March 2013, pp. 332–341.

[9] T. W. Schiller and M. D. Ernst, "Reducing the barriers to writing verified specifications," in *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA '12. New York, NY, USA: ACM, 2012, pp. 95–112.

[10] T. D. LaToza, W. B. Towne, C. M. Adriano, and A. van der Hoek, "Microtask programming: Building software with a crowd," in *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '14. New York, NY, USA: ACM, 2014, pp. 43–54. [Online]. Available: http://doi.acm.org/10.1145/2642918.2647349

[11] K. T. Stolee and S. Elbaum, "Exploring the use of crowdsourcing to support empirical studies in software engineering," in *International Symposium on Empirical Software Engineering and Measurement*, 2010.

[12] D. Liu, R. G. Bias, M. Lease, and R. Kuipers, "Crowdsourcing for usability testing," *Proceedings of the American Society for Information Science and Technology*, vol. 49, no. 1, pp. 1–10, 2012.

[13] S.-H. Kim, H. Yun, and J. S. Yi, "How to filter out random clickers in a crowdsourcing-based study?" in *Proceedings of the 2012 BELIV Workshop: Beyond Time and Errors - Novel Evaluation Methods for Visualization*, ser. BELIV '12. New York, NY, USA: ACM, 2012, pp. 15:1–15:7. [Online]. Available: http://doi.acm.org/10.1145/2442576.2442591