# Debugging with the Crowd: a Debug Recommendation System based on Stackoverflow

Martin Monperrus, Anthony Maia

# Debugging with the Crowd: a Debug Recommendation System based on Stackoverflow

Martin Monperrus, Anthony Maia

University of Lille & Inria

Technical report #hal-00987395, INRIA, 2014

## ABSTRACT

Debugging is a resource-consuming activity of software development. Some bugs are deeply rooted in the domain logic but others are independent of the specificity of the application being debugged. The latter are "crowd-bugs": unexpected and incorrect output or behavior resulting from a common and intuitive usage of an API. On the contrary, project-specific bugs are related to the misunderstanding or incorrect implementation of domain concepts or logics. We propose a debugging approach for crowd bugs, which is based on matching the piece of code being debugged against related pieces of code on a Q&A website (Stackoverflow). Based on the empirical study of Stackoverflow's data, we show that this approach can help developers to fix crowd bugs.

## 1. INTRODUCTION

Debugging is the activity that consists of understanding why a piece of code does not behave as expected. Developers may debug the code they have just written, or an old piece of code for which a user has provided a bug report. The process of debugging a particular bug is deeply related with the nature of the bug. For instance, debugging a segmentation fault means focusing on allocation and deallocation of memory, debugging concurrency errors means focusing on the interleaving of instructions under a particular schedule.

In this paper, we claim that there exists a class of bugs that relate to the common misunderstanding of an API. Those bugs have a common characteristic: they appear again and again, independently of the problem domain of the buggy application. For instance, as witnessed by the myriad of related questions on the web, there are plenty of programmers who experience that `parseInt("08")` returns 0. This piece of code does not behave as expected. We call this class of issues "crowd bugs".

The nature of "crowd bugs" has two direct implications. First, for crowd bugs, it is likely that the crowd bug has already occurred several times and there exists a description of the problem somewhere on the web, along with its explanation and fix. In other terms, the crowd has already identified the bug and its solution. Second, for a programmer who experiences again a crowd bug, she can ask the crowd how to fix the bug. This is what we call "debugging with crowd": giving a piece of code to the crowd for obtaining the fix.

Hence, the nature of crowd bugs (same symptom, independently of the domain) enables a specific kind of debugging (asking the crowd). In this setup, the problem statement of debugging becomes appropriate for a recommendation system: the debugger should match as closely as possible a similar bug and its fix.

This paper contributes in this domain. We present a debugging system for Javascript that is designed as a recommendation system, dedicated for crowd bugs. We build our debugger based on empirical insights obtained on the data from Stackoverflow. Stackoverflow is a popular Q&A website that is dedicated to programming questions.

Our contributions are as follows:

- the definition of "crowd bug" and "debugging with the crowd";
- the empirical analysis of Stackoverflow data with respect to Javascript crowd bugs (which shows that Stackoverflow is not good at taking code as input query);
- the empirical proof that code snippets describing crowd bugs can act as a self-contained and clear description of the bug under analysis;
- the design of crowd debugger that automatically relates a breakpointed bug and the solution provided by the crowd.

The remainder of this paper is as follows. Section 2 defines the notion of "crowd bugs". Section 3 is an empirical study of crowd bug occurrences on the Q&A website Stackoverflow. Section 4 shows that code snippets can act as debug queries. Section 5 describes the design of a crowd-based debugger. Section 6 discusses the related work.

## 2. CROWD BUGS

To define what a "crowd bug" is, let us start with an example. In Javascript, there is a function called `parseInt`, which parses a string given as input and returns the corresponding integer value. Despite this apparently simple description and self-described name, this function poses problems to many developers, as witnessed by the dozens of Q&As on this topic on Q&A websites[1] (this is likely an underestimation since we believe that many developers google the issue before asking a question on a mailing list or a Q&A website). Many Q&As relate to the same issue: *Why does `parseInt("08")` produce "0" and not "8"?*.

The answer is that if the argument of `parseInt` begins with 0, it is parsed as octal. Why is the question asked again and again? We hypothesize that the semantics of `parseInt` is counter-intuitive for many people, and consequently, the very same issue occurs in many development situations, independently of the domain.

### 2.1 Definition of "Crowd Bugs"

We define a "crowd-bug" as follows.

*Definition: A "crowd-bug" is an unexpected and incorrect output or behavior resulting from a common and intuitive usage of an API.*

On the contrary, project-specific bugs are related to the misunderstanding or the incorrect implementation of domain concepts or logics. Since those domain-independent bugs occur many times across different teams and projects, we call them "crowd bugs".

### 2.2 Characteristics of Crowd Bugs

*Where do crowd bugs come from?* Crowd bugs appear when a developer of an API makes design decisions that go against the common sense, "common sense" being defined as the intuition and expectation being hold by many developers. In the `parseInt` example, most people expect that `parseInt("08")` returns "8".

We see two main reasons behind crowd bugs. First, the API under consideration may seem comprehensible enough for developers for *not reading* the documentation. Second, the API may assume something *implicitly*. For instance, `parseInt` *implicitly* assumes that prefixing the string with "0" means that an octal basis is chosen. There exists a second version of `parseInt`, which takes a radix as second parameter. If the short and implicit form with one parameter had been forbidden, this crowd bug would have likely never existed. A detailed study of why crowd bugs appear is out of scope of this paper, it requires inter-disciplinary work between different fields such as software engineering and psychology.

*How to fix crowd bugs?* To fix this kind of bug, the developer first needs to be sure that the observed incorrect behavior is not domain-specific. Once all domain-specific explanations are discarded, the common and sensible way to fix the bug is to search for similar problems over the internet. Since crowd bugs are not related to domain-specific knowledge and logics, they can be debugged by the crowd itself. The description of the symptom is enough for other developers to recall the occurrence of the same bug and to give the fix.

> To fix a "crowd-bug", ask the crowd.

For example, on the Q&A site Stackoverflow, all questions related to this particular unintuitive point of `parseInt` are answered. The most voted answer[2] has been given only one minute after the question has been posted.

Crowd-bugs are not really studied in the literature. Only Carzaniga et al. proved that the web contains some descriptions of "crowd bugs" [3]. Our motivating example about `parseInt` showed that Stackoverflow also contains such bugs. Actually, many instances of Carzaniga et al.'s bugs can be found on Stackoverflow as well.

## 3. AN EMPIRICAL STUDY OF CROWD BUGS

In this section, we explore whether the idea of debugging with the crowd makes sense empirically. We pose a number of research questions and answer them based on real data extracted from the Q&A site Stackoverflow.

### 3.1 To what extent does Stackoverflow contain "crowd bugs"?

Beyond the anecdotal examples we have just discussed, we would like to know to which extent Stackoverflow contains "crowd bugs". If there are many answered "crowd bugs" on Stackoverflow, it would make sense use this wealth of information to ease debugging.

However, it is not possible to manually assess whether the millions of Q&As of Stackoverflow refer to crowd bugs or not. Hence, we need an automated technique of finding "crowd bugs".

This technique is meant to be more qualitative than quantitative. What we need to know is whether there are many answered crowd bugs on Stackoverflow. We do not need their exact number or even a precise estimation of their proportion. Knowing that there exist plenty of them is enough to motivate a crowd-based debugger. Consequently, it is not an issue if the identification technique yields some false positives (Q&A that are identified as crowd bugs while they are not) or false negatives (Q&A that are not identified as crowd bugs while they are).

We propose the following criteria to identify Q&As as usable "crowd bug". First, it must contain one snippet in the question since crowd-debugging consists of understanding a piece of code being debugged. Second, they should be limited to a particular language (for sake of analysis). Third, they should have an accepted answer to be sure that the crowd was able to solve the bug.

[1] `http://stackoverflow.com/search?q=[javascript]+title%3AparseInt&submit=search`

[2] `http://stackoverflow.com/questions/850341/how-do-i-work-around-javascripts-parseint-octal-behavior`

With respect to the second point, we focus in this paper on Javascript-related Q&As due to the inspiration of Carzaniga et al. [3]. To select Javascript crowd bugs, we use a filter on the tagging metadata of the question: we select questions if and only if they are tagged by [javascript].

The Dec. 2011 dump of Stackoverflow Q&As[3] contains 7,397,507 Q&As. By applying the 3 filters aforementioned (contains code, javascript, answered), we obtain 70,060 Q&As. *To our opinion, 70,060 is a large number, validating our intuition that there is a wealth of information in crowd bugs on Stackoverflow.* In the rest of this paper, we will refer to those Q&As as dataset D1, "the initial dataset".

> There exists a large number of Q&As that satisfy the our "crowd-bug" criteria.

## 3.2 What is the shape of answered stackoverflow Crowd-Bugs ?

We now study the characteristics of the snippets of the 70,060 crowd-bugs of dataset D1.

In Stackoverflow, code is put in <code> or <pre> HTML tags. First, Stackoverflow users may have split their problems into several snippets (resulting in several pairs of <code> or <pre> HTML tags). This happens when the user thinks it would simplify the problem statement. We have computed the distribution of the number of snippets per initial question of dataset D1. Figure 1 (top) represents the boxplot of the distribution: the median number of snippets per question is 2, which is low. Actually, 29,159/70,060 (41%) of the crowd bugs contain only 1 snippet. In other words, the majority of Stackoverflow users asks questions with no pre-analysis of the problem statement in different snippets.

Second, we have computed the number of lines of those 29,159 single-snippet questions. Figure 1 (bottom) represents the boxplot of the distribution of snippets and lines: the median number of lines is 8 and 75% of those snippets have less than 17 lines of code. This shows that the code describing crowd-bugs is usually short.

It is interesting to know whether we can reason on the abstract syntax of the snippets. For this, we have measured the number of snippets that are parsable with a Javascript parser. In total, 48% (14,087/29,159) snippets are parsable. This relatively high number motivates us to study systems based on the abstract syntax trees of Stackoverflow snippets. Those 14,087 Q&As containing a single parsable Javascript snippet form the $SPJ$ dataset.

> Many "crowd-bugs" are described with a short, parsable code snippet.

# 4. CODE SNIPPETS AS STACKOVERFLOW QUERIES

[3]`http://www.clearbits.net/torrents/1881-dec-2011`, file stackoverflow.com.7z

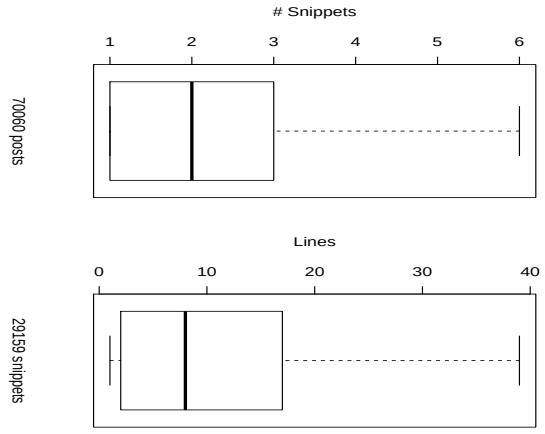


**Figure 1: Descriptive statistics of our Selected Q&As from Stackoverflow**

We have seen that many crowd-bugs are indeed described with code snippets. Our intuition is the following.

> If the fault being debugged is actually a crowd-bug, there should be similar snippets on popular Q&A sites such as Stackoverflow.

"Crowd debugging" means out-sourcing pieces of code being debugged to the crowd. A "crowd-based debugger" selects excerpts of code, sends them to a Q&A web site, and retrieves the posts in which there is a very similar snippet.

## 4.1 What is the efficiency of Stackoverflow in response to code snippet queries?

We would like to know whether the search engine of Q&As websites, such as Stackoverflow, is able to well handle code snippets as input. Feeding a Q&A website with a code snippet simulates a user having a problem, and querying Stackoverflow with the problematic code (such as `parseInt("08")`). Our idea is to feed the Stackoverflow search engine with a code snippet that already exists on Stackoverflow.

We consider that the Q&A website handles well the snippet if the post from which the code comes from is ranked high in the search engine results. This corresponds to the case where a developer posts a snippet describing the same crowd-bug and gets the correct answer.

On the contrary, if the Q&A website does not recognize that there is actually this problem on its own database, this is a piece of evidence that the Q&A website is bad at answering code snippet queries. In this case, a crowd-based debugged would not be very useful and a special indexing technique would need to be invented.

To answer our research question, we have conducted the following experimental process. For each Q&A of a given dataset, we query Stackoverflow with the snippet of the question and we observe the result. There are three possible outcomes.

First, Stackoverflow may consider that the query is invalid: this happens when the snippet query is too long or contains special characters. Second, it happens that Stackoverflow returns no result at all (although the exact text of the snippet comes from Stackoverflow itself). Third, the expected Q&A may be found, and in that case, we collect its rank. Note that Stackoverflow restricts the number of automated queries; consequently, 1000 queries take in average 3 hours.

Eventually, we measure to what extent Stackoverflow is able to rank the Q&A containing the snippet high in its result list. Stackoverflow does not document how they index the questions and whether they specifically handle the content of code snippets. If a majority of Q&As are correctly retrieved with snippets as input, it means that Stackoverflow handles well code snippets.

Out of the 70,060 of dataset D1 aforementioned Stackoverflow, we have randomly extracted 1000 posts and their respective code snippet. We call this dataset D2, it is available upon request. For the corresponding 1000 queries to Stackoverflow, the results are as follows:

- 377/1000 (38%) snippets are considered as non-valid queries.
- 374/1000 (37%) snippets yield no results (the expected Q&A is not found).
- 146/1000 (15%) snippets yield a perfect match (the expected Q&A is ranked #1).
- 235/1000 (23% incl. the 15% below) snippets yield a good match (the expected Q&A is ranked $\leq 10$).
- 14/1000 snippets yield a good match at rank $> 10$.

To us, those results are surprising. First, although we give verbatim content taken from Stackoverflow as query, $38 + 37 = 75\%$ of the queries are not answered at all. This is a large number showing that Stackoverflow is not good at taking code as input query.

Second, for those snippets where the Q&A is found, Stackoverflow is not able to rank the corresponding Q&A as #1. This means that there are many similar Q&As for this particular input: this is again a piece of evidence regarding the existence of crowd bugs (many instances of the same bug are posted).

*Stackoverflow is not good at handling code snippets as query.* This motivates us to: 1) improve the performance of crowd-bug matching by preprocessing the code snippets; 2) build a dedicated index that would yield better results.

## 4.2 Improving Crowd Bug Search

We have seen in 4.1 that Stackoverflow's search engine is not really good at handling code snippets as input. Let us now discuss ways for improving snippet-based search on Stackoverflow. First, we explore the efficiency of pre-preprocessing techniques to improve the response of Stackoverflow. Second, we study how to build our own index, a specific index dedicated to handling code snippets of crowd-contributed Q&As.

Let us first briefly explain what is indexing and how this matters in our context. Indexing provides users with a real-time search experience over a database of millions of Q&As. The standard technique of indexing-based search consists of two transformation functions $t_{index}$ and $t_{query}$. The first function $t_{index}$ transforms the documents (the Q&As) into terms, and the second function $t_{query}$ transforms the query into terms as well. The ranking consists of matching the document terms against the query terms, possibly with some weighting strategies. $t_{index}$ and $t_{query}$ are not necessarily identical, since the nature of documents (style, structure, etc.) is often different from the nature of queries.

The results discussed in section 4.1 actually proves that Stackoverflow uses different $t_{index}$ and $t_{query}$ since 75% queries made with an exact fragment taken in the text of Q&As yield no result at all. This is not a bug, as explained above, there are many reasons for which $t_{index}$ and $t_{query}$ may be different. This result only shows that Stackoverflow's search engine has not been designed to handle code-based queries.

### 4.2.1 Improving Crowd-based Debugging By Preprocessing Code Snippets

The poor results of Stackoverflow is due to a misalignment between $t_{index}$ and $t_{query}$ with respect to code-based queries. Our idea is to introduce a pre-processing $pp$ function in order to reduce the misalignment. This function takes a code snippet as input and returns a string as output that is appropriate for a query to Stackoverflow.

That is, $t_{query} \circ pp$ should produce terms that better correspond to terms produced by $t_{index}$ of Stackoverflow. This is directly measurable: the use of $pp$ should decrease the number of not valid and not found Q&As.

*How efficient is it to pre-preprocess Javascript code snippets?* We have experimented different pre-processing functions, we present here the ones that best improve the capability of Stackoverflow to respond to code-based queries. The first pre-processing function is called $pp_1$, it matches all alphanumerical sequences in the snippet, with the regular expression "[0-9a-zA-Z_$]+". As a side effect, this pre-processing function removes all punctuation characters. The second pre-processing function is called $pp_2$ and works in a more semantic way, it parses the Javascript snippet, and returns the list of values contained in abstract syntax tree nodes (AST nodes) representing names (e.g. variable names) and string literals.

With those two pre-processing functions, we follow the same evaluation process as described in Section 4.1. For the 1000 Q&As, we pre-process the snippets the of the question, and then query Stackoverflow. We then observe whether the source Q&As is given in the results (and at which rank). Table 1 gives the results of this process. In this table, SOF refers to $StackOverFlow$, hence $t_{index} = SOF$ means that the indexing function is that of Stackoverflow. Both pre-processing functions enable us to find more Q&As (there are less "not valid" and "not found"). They also both improve the average ranking of the perfect answer.

| | Not-valid | Not-found | Rank#1 | Rank≤10 | Rank≤250 |
|---|---|---|---|---|---|
| $t_{index} = SOF$, $t_{query}$=Raw | 337 | 374 | 146 | 235 | 289 |
| $t_{index} = SOF$ ,$t_{query}$=$pp_1$ | 308 | 338 | 208 | 288 | 333 |
| $t_{index} = SOF$ ,$t_{query}$=$pp_2$ | 343 | 338 | 208 | 290 | 324 |
| $t_{index}$=$pp_2$, $t_{query}$=$pp_2$ | 5 | 130 | 511 | 758 | 865 |

**Table 1: Using Stackoverflow Snippets as Code-based Queries (SOF refers to *S*tack*O*ver*F*low, hence $t_{index} = SOF$ means that the indexing function is that of Stackoverflow, "Raw" means that no pre-processing is used)**

Of the two pre-processing functions, the one that is based on the AST ($pp_2$) is slightly better. This is due to the fact that the terms produced by $pp_1$ also contains language keywords (e.g. "for" or "if"), which somehow distort the matching against document terms. The AST based pre-processing ($pp_2$) does not suffer from this issue.

> Preprocessing code snippets that describe crowd bugs is able to improve crowd-based debugging.

#### 4.2.2 *Improving Crowd-based Debugging with a Specific Index*

Now that we have explored a better $t_{query}$, let's study the effect of $t_{index}$. We build our own index of the Stackoverflow data and use again the evaluation process described in Section 4.1 and 4.2.1. We use an off-the-shelf indexing technology called Lucene[4] for this. We use the same function for $t_{index}$ and $t_{query}$: $pp_2$, because that was the champion according to the experiment presented in Section 4.2.1.

*What if one aligns $t_{index}$ and $t_{query}$?* The last row of Table 1 gives the result. Out of 1000 queries, only 135 are not answered within the first 250 results, and 511 snippets are ranked first. Compared to the first row (basic query to Stackoverflow), this is a real improvements: unanswered queries drop from 75% to 13%, top rank queries increase from 14.6% to 51.1%.

> Building a specific index of Q&As by aligning $t_{index}$ and $t_{query}$ yields a real improvement of crowd-based debugging.

If a Q&A website such as Stackoverflow wants to support code snippet queries $pp_2$ is a good candidate for $t_{index}$ and $t_{query}$.

### 4.3 Summary

We have shown in this section that Stackoverflow does not well handle code queries as input. This is not surprising since StackOverflow is designed for mostly handling text queries. However, we have also shown is that one can 1) improve crowd-based debugging by preprocessing the code snippets; 2) or build a dedicated index to obtain even better results. This is will be used in the design of a crowd-based debugger that we will present in Section 5.

[4] `https://lucene.apache.org/`, last accessed Feb 26 2013

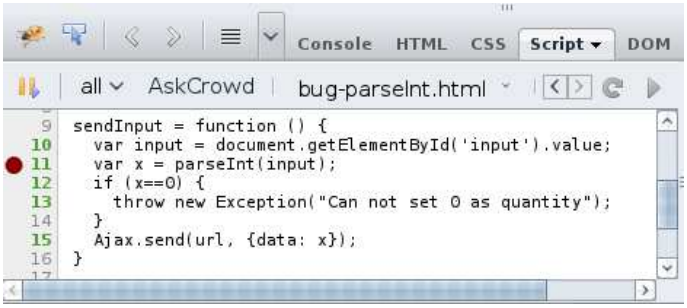


**Figure 2: A screenshot of our prototype crowd-enhanced Javascript debugger. The button AskCrowd selects the snippet surrounding the breakpoint in order to quickly find an answer (the breakpoint is a red circle at line 11 in the left hand side margin of the program).**

## 5. DESIGN AND PROTOTYPE OF A CROWD-BASED DEBUGGER

We know that: 1) Q&As websites such as Stackoverflow contain Q&As related to generic crowd bugs; 2) those Q&As often contain a snippet that describes the bug; 3) it is possible to achieve a good accuracy when matching code snippets against each others.

Consequently, we propose a crowd based debugger as follows. The developer sets a breakpoint on the line of the crowd bug and then asks the crowd by clicking on the button "Ask the crowd". The debugger extracts the $n$ lines around the breakpoint and preprocesses them with a function $f$. This results in a query $q$. The debugger then sends the query to a server and retrieves a list of potential answers to the problem. Those potential answers are those for which the snippet of the question closely matches the snippet of the query. Figure 2 shows a tentative user interface for such a debugger.

We have implemented such a prototype system for Javascript. The prototype extends Firebug, a Javascript debugger for Firefox. For the `parseInt` example, the developer retrieves the perfect answer in just one click. This prototype system decreases the time of debugging crowd bugs. The debugger is only parametrized by $n$, the number of lines taken around

the breakpoint. The optimal value of $n$ depends on the programming language. In our implementation we set it to 8[5], the median size of code snippets describing crowd-bugs on Stackoverflow (see Figure 1 in Section 3.2).

> A crowd-based debugger extracts some pieces of code and runtime state from the bug under analysis and retrieves relevant information from a database of crowd provided information.

# 6. RELATED WORK

Barua et al. [2] analyze the topics of Q&As on Stackoverflow. Their analysis has a different scope from ours: they do not discuss the nature of the questions, they do not put Q&As in action as we do.

Hartmann et al [4] invent a recommendation system for fixing compiler errors. Their system is fed with monitoring data. Kallenbach [5] extends the system for runtime errors in Ruby. Contrary to setting up a database of specific data, our system is fed with data obtained from Q&As looked up in websites.

Treude et al [6] present an empirical study of Stackoverflow Q&As. Their key result with respect to our debugging system is that question with code snippets are more often answered that others. In other words, the Stackoverflow is already driven by code snippets and the Stackoverflow data base continuously includes more and more crowd bugs.

Zagalsky et al. [8] also focus the intrinsic value of code snippets of Q&As websites such as Stackoverflow. They use those snippets to build a search engine dedicated to code snippets. We use in a different context: at debugging time, integrated with breakpoints.

Carzaniga et al. [3] also handle Javascript bugs. However, they handle the bugs from the perspective of the user who is provided with alternatives in order to complete some requests. On the contrary, we provide alternative recommendations to the developer who has set up a breakpoint in her code.

Bacchelli et al. [1] are among the first to set up an integration of Q&As data within the IDE for sake of productivity. Their system enables developers to browse and comment Q&As in tight integration with the source code under development. We are along the line of integrating Q&As data in the IDE. Bacchelli et al. do not consider involving the crowd in the debugger directly for debugging activities.

Popescu and colleagues explicitly bridged the intelligence of the crowd with error diagnosis and repair [7]. They target the understanding of compiler errors of Java programs. We address another kind of errors: runtime errors in Javascript. Consequently, their prototype system is integrated into an IDE (BlueJ) while ours is integrated into a debugger (Firebug).

[5]Future work will study the influence of the value of $n$ on the results. This requires setting a realistic corpus of client code that contain crwod bugs. Creating such a corpus is a hard problem.

# 7. CONCLUSION

In this paper, we have presented a new approach to debugging. This approach states the debugging problem as a recommendation system problem. We show that there exists a class of bugs, called "crowd bugs", for which our debugging approach is relevant. We present an empirical study showing that Stackoverflow is not good at taking code as input query. Then, we design a new debug system founded on empirical insights obtained from 70,060 Q&As extracted from Stackoverflow. This system embeds the crowd of a Q&A website such as Stackoverflow in the debugger directly. Future work will explore whether the crowd-provided fixes can be automatically assessed and integrated, without the need for human intervention.

# 8. ACKNOWLEDGMENTS

We would like to thank Maria Gomez Lacruz and Luca Ponzanelli for their valuable feedback on this paper.

# 9. REFERENCES

[1] A. Bacchelli, L. Ponzanelli, and M. Lanza. Harnessing stack overflow for the ide. In *Proceedings of the Recommendation Systems for Software Engineering*, pages 26–30, 2012.

[2] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, pages 1–36, 2012.

[3] A. Carzaniga, A. Gorla, N. Perino, and M. Pezzè. Automatic workarounds for web applications. In *Proceedings of the 2010 Foundations of Software Engineering Conference*, pages 237–246, 2010.

[4] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer. What would other programmers do: suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1019–1028, 2010.

[5] M. Kallenbach. Crowdsourcing suggestions to programming problems for dynamic, interpreted languages. Master's thesis, RWTH Aachen University, 2011.

[6] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web? In *Proceedings of the 33rd International Conference on Software Engineering*, pages 804–807, 2011.

[7] C. Watson, F. B. Li, and J. Godwin. Bluefix: Using crowd-sourced feedback to support programming students in error diagnosis and repair. In *Advances in Web-Based Learning - ICWL 2012*, pages 228–239. Springer Berlin Heidelberg, 2012.

[8] A. Zagalsky, O. Barzilay, and A. Yehudai. Example overflow: Using social media for code recommendation. In *International Workshop on Recommendation Systems for Software Engineering (RSSE)*, pages 38–42, 2012.