Automatically locating relevant programming help online

Oleksii Kononenko, David Dietrich, Rahul Sharma, and Reid Holmes
School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
Email: okononen, d4dietri, r64sharm, rtholmes{@uwaterloo.ca}

***Abstract*—While maintaining software systems, developers often encounter compilation errors and runtime exceptions that they do not know how to solve. Solutions to these errors can often be found through discussions with other developers on the Internet. Unfortunately, many of these online discussions do not contain relevant answers. We have developed an approach to automatically query and analyze online discussions to locate relevant *solutions* to programming problems. Our tool, called Dora, is integrated into Visual Studio and allows developers to query and evaluate solutions within their development environment, enabling them to reduce context switching between their development tasks and their search sessions. We have performed a semi-controlled experiment to validate the utility of our search approach with 18 tasks, finding that our approach provides 55% more relevant results than traditional web searching approaches.**

I. Introduction

Developers frequently wonder, "How did this runtime state occur?" [1]; this question often arises as a result of an exception or other programming error. Given the vast nature of software systems, developers sometimes encounter errors that are new to them that they do not know how to solve. Determining how to resolve these errors often involves investigating documentation specific to the software being worked on [2], discussing the error with fellow developers [3], or searching the Internet and investigating various online resources in the hope of finding a suitable solution or explanation [4], [5].

Identifying helpful programming information online can be challenging; while developers often post their problems online, many of these posts have no solution at all, or the provided solution is incorrect. The volume of unhelpful information can cause a developer to investigate many unsuitable posts before finding the correct answer. Time spent investigating unhelpful posts increases the duration of search sessions, as well as increasing likelihood that the developer abandons the search task altogether [6].

We have designed an approach to help developers identify posts from discussion forums that are contextually relevant to the developer's problem. Once a search is initiated from our tool (called Dora) within the developer's Integrated Development Environment (IDE), several common programming-related online resources are searched. The results of these searches are evaluated according to our model that captures the properties of helpful programming posts. The most relevant results are displayed to the developers within their IDE, enabling easy result investigation without losing the context on their current development tasks.

We have evaluated our model that describes helpful programming posts as well as the complete Dora tool. The model was trained with 255 results that were returned from 14 queries. As a result of this training evaluation, we identified six characteristics that capture relevant aspects of of a helpful programming post.

Dora was evaluated through a semi-controlled experiment using 18 search tasks. The tool was compared to equivalent queries performed with Google[1] and Stack Overflow[2], a frequently-used source of development questions [7]. We found that Dora provides a higher proportion of relevant results (47%) than Google (30%) or Stack Overflow (26%). Dora also returned relevant results earlier in the result list (mean 1.0) than Google (2.83) or Stack Overflow (2.0).

The contributions of our paper are:

1) We developed a model that identifies online posts likely to be useful for answering programming questions.
2) We defined a set of characteristics that evaluates whether textual material matches our model and performed a validation of the utility of these properties.
3) We implemented a tool that embodies the matching characteristics and a validation of its effectiveness.

The remainder of the paper is as follows. A motivating scenario is provided in Section II. Related work is covered in Section III. The model, its weighting, and its relevance levels are presented in Section IV. Dora and its evaluation is described in Section V. Discussion is included in Section VI; Section VII concludes.

II. Scenario

In this section we will give a brief overview of how a developer would use Dora in practice. John is a software developer who uses the Visual Studio IDE to maintain a system written in C#. While developing a new feature, he encounters a compilation error that states "`Bad array declarator`" in the Error view (Figure 1(A)). John has never seen this error before, and the error message alone is not enough for John to figure out what he needs to do to fix the error and get

[1]http://google.com
[2]http://stackoverflow.com

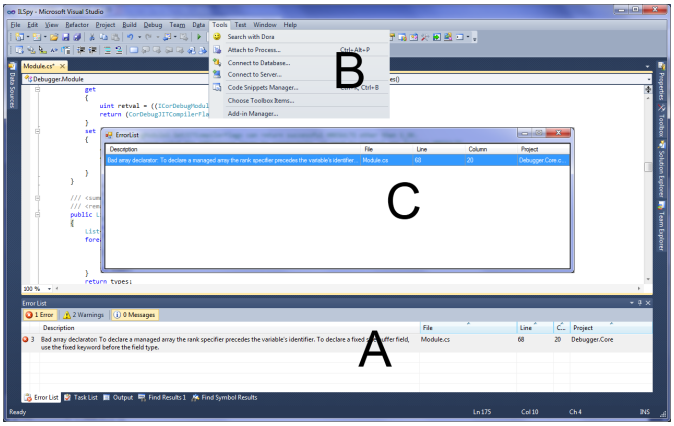

Fig. 1. The Visual Studio IDE with a compiler error.

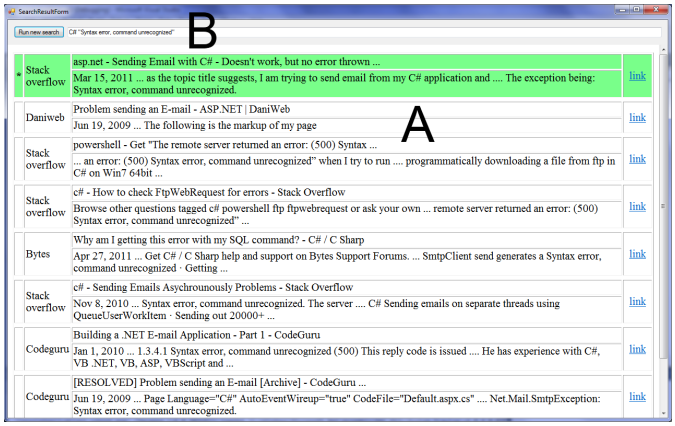

Fig. 2. The results of our automated search.

back to his task. To solve this problem, John switches from the IDE to his web browser and manually enters in a query to a search engine to try to see how other developers resolved this problem. Unfortunately for John, several of the returned results have developers asking about this problem, but do not contain solutions.

In contrast, when using Dora, John initiates the query within the IDE (Figure 1(B)). He selects the error that he would like to find solutions for (Figure 1(C)); Dora then generates a query automatically and searches relevant online programming forums. Dora evaluates the results returned from these forums to identify posts that are likely to help resolve John's problem. The results of this query are displayed within Visual Studio (Figure 2); each result includes a page title, a short description and the link to the page (Figure 2(A)). The results are sorted according to their utility for John's query; irrelevant posts or those that do not contain solutions are elided. If John chooses to modify his query, he is able to manually edit the search terms and perform the search again (Figure 2(B)).

After John has resolved his compilation errors, he is able to run his code. Unfortunately, he encounters a runtime exception (`AssemblyResolutionException`) when calling an external library. He again initiates a Dora search, and is able to quickly find a solution without leaving the IDE or having to investigate any dead-end forum posts. By clicking on the link of an appropriate result, John can view the full thread within the Dora IDE view.

III. Related Work

Understanding programming errors (and how to fix them) is a fundamental software development skill. Compilation errors are one kind of programming problem developers must resolve. Unfortunately, novice developers often have trouble understanding compilation errors [8], [9]. Nienaltowski et. al. [9] also found that experienced developers are able to more quickly solve compilation problems indicating that novice developers could benefit from learning from others.

Hartmann et. al. [10] developed the HelpMeOut system to assist developers by automatically suggesting solutions to compiler errors. HelpMeOut provided a structured environment that stored and provided suggestions on compilation errors for developers. While the approach considered solutions gathered from other users, it did not query the Internet for these resources. In contrast, Dora is a lighter-weight approach that does not maintain its own database, although HelpMeOut results are more structured and succinct than those returned by Dora.

Experienced developers often leverage the past work of other developers while developing their systems (e.g., [11]). This kind of behaviour has been demonstrated to be effective in practice [12]. In particular, a number of software recommendation systems have been created to help developers accomplish tasks by looking at the examples from existing systems (e.g., CodeFinder [13], Jungloid mining [14], and Strathcona [15]).

Previous work by Parnin and Treude [16] on using crowd documentation for application programming interfaces (APIs) has shown the benefits of using social media as a means of documenting APIs. In particular, their work focused on API documentation for the JQuery library and found that by using Stack Overflow alone 84.4% of all APIs was covered, while official forums contained documentation for only 37.0% of the API. A more recent report by Parnin et. al. [17] explored three additional API's: Google Web Toolkit, Java and Android. This report confirms the previous work showing that in the case of the Android API, 87% of the API has been covered by at least one post on Stack Overflow. Although this work did not directly look at errors, the results show that collaborative online resources can contain valuable programming information.

Bacchelli et. al. [18] have developed Seahawk, an Eclipse plugin that integrates Stack Overflow with the Eclipse IDE. Seahawk tightly integrates the IDE and Stack Overflow enabling developers to query, view, link, and incorporate examples from Stack Overflow into the their system. Dora is complementary to Seahawk; the primary focus of our tool is to improve the relevance of the returned results from multiple online resources.

IV. IDENTIFYING HELPFUL THREADS

Developers frequently post questions about programming problems online. Conversational threads are composed of posts that contain questions, answers, discussion, code snippets, and other conversation. When searching for help online, developers want to find threads about their problem containing posts that answer their questions as quickly as possible. Unfortunately, developers frequently encounter online discussion threads that contain only questions (and no answers), or questions that have incorrect answers. We have identified several characteristics that model posts that are likely to solve programming problems; these characteristics have been implemented in Dora.

A. Characteristics of helpful posts

The primary challenge is to take a textual document (e.g., a post on a web page) and determine whether it provides a potentially-successful solution to a given query. To address this problem, we identified eight characteristics, or properties, that we hypothesize will model potentially useful posts. Although each of these characteristics is weighted equivalently (with a score of one), some of them are able to score more than once (these are clearly identified below).

1) Marked solved/answered: Online development resources often provide the ability for developers to flag threads as having correct solutions (as well as specific posts for providing the right solution). This marker (usually displayed prominently near the top of the thread and near the correct solution) can be helpful to quickly identify whether a post is helpful. From a developer's point of view, the presence of this marker suggests that there should be a helpful result for them within the included comments (as long as the post is relevant to their query).

2) No replies: It is common to encounter threads which have no replies to the original question. These kinds of posts represent dead threads that provide no benefit to the developer. While the date of the post could be considered (e.g., a post made only five minutes ago not having a solution may not be surprising), we treat all posts with no replies equivalently.

3) N replies: For threads that have one or more replies, each reply is counted. Hence for N replies, the total value assigned to the thread would be N. The justification behind this is that more replies may indicate increased discussion about the problem or the presence of alternative solutions to the error.

4) N replies contain source code: As compilation errors generally arise due to source code problems, another positive characteristic are responses that provide code snippets. It is also common that runtime exceptions might occur because of logical errors in source code or incorrect object initialization. Source code in these kinds of discussion forums are predominantly placed in easily-identifiable HTML markup. We search for this HTML markup to locate source code snippets in a replies. The presence of source code in a reply post can indicate that a developer is trying to provide a concrete solution to the original question; we assign a value of 1 to each reply that contains a source code snippet.

5) N author replies: When developers reply to their own threads, they are actively encouraging greater discussion on their problem. Authors also post replies to their own posts when other developers request additional information or clarification that could be helpful for resolving the original problem. We apply a value of 1 for each reply made by the initial author of a thread.

6) Author made last reply: We also identify posts where the author made the final comment on their question. This measure is included because developers often close their posts by thanking one of the other developers on the thread for providing a solution.

7) Finding positive themes using Sentiment Analysis: The positive or negative attitudes present in the thread can also give insight into the influence of the solutions provided by other developers. This method of analyzing one's attitude is called Sentiment Analysis [19]. We use the Alchemy API Sentiment Analysis tool[3] to analyze the sentiment of replies to the question asked. We split sentiment analysis into two characteristics: replies from the author, and replies from other developers. Our justification for this is that positive replies from the author may indicate that they are expressing gratitude or confirming that everything now works whereas negative replies from the author may indicate that the posted solutions have not worked. Other developers may add that the provided solution has worked for them, or to chime in that they are also unable to solve this problem.

B. Weighting the model

The model characteristics above were identified by the authors based on their experience with online help sources. While we do not claim that they are comprehensive, we believe they capture many salient properties of helpful programming threads. It is natural to assume that the value of these characteristics are not equal. While our model applies equal weights to each characteristic (with the exception of those that can be counted more than once), it is necessary to weight the characteristics accordingly to assess the utility of any given result.

1) Experimental approach: We performed a semi-controlled experiment to identify a candidate set of weights for our characteristics. During the experiment, we performed 14 searches using our tool that returned 264 search results (threads). The searches presented compilation errors and runtime exceptions chosen by the authors based on their experience to represent a reasonably diverse set of exceptions. The first three authors (participants) independently evaluated these search results and assigned a value on a scale from 1 to 5 to capture the relevance of each result. A value of 1 indicates that there is no useful information to be found in the thread, while the value 5 means that the thread provides an exact solution to the query. 9 out of 264 results were discarded because the participants did not sufficiently agree on the relevance of the result (e.g., the difference between

[3]http://www.alchemyapi.com/api/sentiment/

at least two assigned values was more than 2). For the remaining 255 results, we averaged the values we assigned to compensate for possible minor differences in how we evaluated the relevance of the thread. We used WEKA [20], a machine learning tool, to apply multiple linear regression to the collected data to derive candidate weights for the remaining 255 data points. The choice of using multiple linear regression was determined by our interest in finding approximation of relevance, rather than classifying each result to a specific bin. The data for all experiments described in the paper are available online.[4]

2) Identifying model weights: The resulting weights from the linear regression are shown in Table I. The value of the baseline is 1.93; this means that every query starts with this value and the scores of each matching characteristic are added to this starting value. The positive values of weights show that a characteristic correlates with a positive impact on overall relevance, while a negative value shows that the associated characteristic correlates with a negative impact. The greater the score, the more relevant the result.

Characteristic	Weight
Marked solved/answered	0.71
No replies	-0.81
N replies	0.06
N replies contain source code	0.01
N author replies	-0.06
Author made last reply	0.20

TABLE I
FINAL WEIGHTS ASSOCIATED WITH EACH MODEL CHARACTERISTIC.

The negative weight for the `Author replied` characteristic conflicts with our initial expectation that the more the author replied, the more useful the thread is. Qualitatively reexamining the threads we scored, we feel the negative correlation arises because (a) authors often reply to say that the solutions suggested by others did not work, (b) authors reply to provide more clarifications to an initially incomplete question, and (c) some threads contains only author's replies (authors often 'bump' their question to make it appear at the top of the question queue).

We do not report weights for Sentiment Analysis because we found that these characteristics create more noise than meaningful contribution to the overall model. For example, the Sentiment Analysis often returned false negative results when an author talked about problems related to a question and naturally used some "negative" words (e.g., fatal execution or memory corruption), or when a poster used phrases like "understood my error" or "I see what was wrong". In such cases, Sentiment Analysis punished the search result even though the question was answered successfully. There were also many false positive values when an author just thanked a developer for a reply, or when other people ended their answer with phrases like "hope it will help you". Code snippets also caused problems; for instance, `boolean success = run();` would be scored positively (because of the word "success"), even though it has no bearing on the effectiveness of the code snippet. For these reason, we removed Sentiment Analysis from our model.

It can be seen that the two largest factors determining the relevance of a post are whether the post has been marked relevant (+0.71) and whether the thread is devoid of any answers at all (-0.81). Interestingly, the author making the last reply has a positive value (+0.20) even though an author being highly active in a thread has a negative value (-0.06 for each subsequent reply). Including source code in a post a fairly minor factor (+0.01); we believe this may be because source code tends to be present in most posts, regardless of whether they are helpful or unhelpful.

C. Determining result thresholds

Since our model assesses the relevance of a result, we wanted to return only relevant discussion threads to the developer. To do this, we classify the returned results as "good", "ok", and "poor"; we never return "poor" results to the developer. In Figure 2, "good" results have a green background and a star sign in the first column, while "ok" results have a plain white background. By differentiating the "good" and "ok" results, developers gain a greater sense of Dora's confidence in a result, beyond its rank in the result view.

To determine the thresholds we re-ran all of the queries from Section IV-B with our derived weights. The thresholds were identified by comparing the weighted score returned by our model to the relevance values assigned by the participants. The participants did not have to reevaluate the relevance of any result; since they evaluated every result of the unweighted model, we were able to retroactively apply the weights without them having to reevaluate the results.

We analyzed our 255 results as one set by ordering the results along the X-axis by their returned score. We found a clear split between "ok" and "poor" results at a score of 2.7; the data showed a knee effect around this point. Because the main contributor to this effect was the solved/answered characteristic, we analyzed the search results below this threshold value to learn how this bias may impact those sites that do not support this kind of metadata. From our data we observed that 197 out of 255 search results (77%) were from sites that support Solved/Answered mark indicating that heavily weighting these results will be unlikely to cause an unfair bias.

The distinction between "good" and "ok" results was more subjective; however, since both "ok" and "good" results would all be returned to the developer, and the order would not be impacted by our choice, this split was somewhat less critical. The data also exhibited a knee effect around 3.25, which led us to select that value as the cutoff between the "good" and "ok" results.

With these weights applied, 8 of the 14 training queries would have at least one "good" result, while 13 of the 14

[4]http://dora.googlecode.com

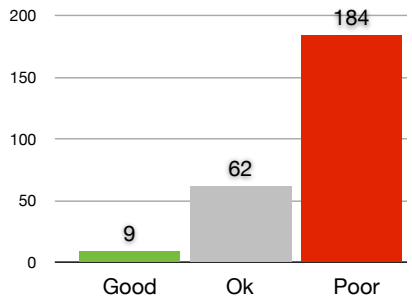

Fig. 3. Number of results in each of the quality categories for the 255 results.

queries had one or more "ok" results (minimum 1, average 4.4). One query had only one "good" result and no "ok" results. The number of results in each quality category for our 255 results is shown in Figure 3.

V. Dora solution search tool

We built the Dora search tool to help developers identify relevant programming posts. Dora was built as a Visual Studio add-in enabling developers to maintain their task context by forming queries and investigating results without having to use external searching tools. This allows them to remain more focused on their tasks and relieves them from constant task switching whenever they need to perform a search [21]. To enhance the performance of the tool, we implemented the search logic on the server side and used the client side only to generate the query string and to provide GUI to examine results. Dora is available for download.[5]

The current Dora prototype searches five specific web sites for solutions:

- Stack Overflow [http://stackoverflow.com]
- Daniweb [http://daniweb.com]
- Bytes [http://bytes.com]
- Codeguru [http://codeguru.com]
- Dev Shed [http://forums.devshed.com]

These five sites were chosen based upon the authors' programming experience. Three of the authors had more than one year of industrial programming experience and found these sites to be the most commonly used when searching for programming help for the C# language.

While our model is not tied to any specific online resource or programming language, there are common properties of these sites that may have influenced the helpful characteristics we identified. For example, the selected sites focus on a thread-based format that enables developers to collaborate, comment, and add metadata to the provided solutions. This contrasts to more static web resources, where commenting is not enabled, or even blog-like posts where comments appear at the bottom but are often not an integral part of the post itself.

We use a set of site-specific regular expressions to extract data from the results returned for each site. For each thread, we delineate all of the posts and extract the author and their message (while deleting any quoted material), as well as checking for presence of the Solved/Answered metadata and the presence of code snippets. Adding new sites is generally straight-forward; for the five sites we support, none requires more than 12 lines of parsing code. While the users cannot add new sites directly to the tool, once they are added to the server side they are automatically available to all clients.

A. IDE integration

Visual Studio provides a view that lists compilation errors whenever they occur. Dora only uses the description field from this view when formulating its queries. Retrieving runtime exceptions is more problematic: we subscribe to Visual Studio's event stream to capture `AppDomain.UnhandledException` events, which occur whenever a runtime exception is encountered. We casted all runtime exceptions to their base class `System.Exception` and took its "Message" field value. Again, only the exception description is used to formulate the query.

To initiate the query, a user has to select the entry for Dora from the Tools menu of Visual Studio; queries are initiated in the same manner for both compilation errors and runtime exceptions. After performing queries to the five sites listed above, the results are analyzed and displayed in another Visual Studio view. An example is shown in Figure 2. Each returned result includes the title of the matched post, a short snippet of the description, an indication of relevance as assessed by our model, and a link to the post itself. To help developers stay focused on their tasks, Dora supports showing the complete text of a particular query result within the IDE. The developer can easily switch from a list of query results to the web view of a result by clicking on the link; this opens the result in a new tab of the results view.

B. Performing searches

Dora automatically generates a query for each of the five programming help sites based on the element the developer queried upon. We manually identified an effective query string through trial and error. While we initially created very specific queries, we noted that including too much contextual information (e.g., information specific to the developer's system) overly restricted results. Ultimately, we settled upon only using the error message; all of the machine specific portions of this message are removed (file names, variables, etc.). The query string was quoted so that the entire string is searched for instead of each word within the query.

We searched each of the five sites separately using the Google Custom Search API[6], rather than querying the site directly. For each site, Dora only examines the first 10 results. Queries to the various sites are performed in parallel to increase performance. Dora's most time consuming action is downloading the HTML for each result for further analysis; the analysis itself is quick enough to be insignificant. Since the current prototype waits for all results before showing them to

[5]http://dora.googlecode.com

[6]http://code.google.com/apis/customsearch/

a developer, the overall waiting time depends on the slowest site. While performance could be improved by evaluating results as they are identified, we have not implemented this streaming behaviour because we want to maintain a stable ordering within the result view.

C. Evaluating the model and tool

Software developers currently use several different approaches to search for programming help information on the Internet. Ultimately, the main question is whether Dora can more effectively identify relevant results than a traditional Internet search. We performed an evaluation to answer the following two specific questions:

RQ-1: Does Dora provide more relevant results than traditional search approaches?

RQ-2: Is the index of the first relevant result better with Dora than traditional search approaches?

The first three authors were the participants in this evaluation. We compared Dora to a Google web search and a search of Stack Overflow using their built in search engine. Google was chosen as it is the most popular general purpose search engine; Stack Overflow was selected as it is currently the most popular and active general programming help forum.

D. Method

We set out to answer **RQ-1** and **RQ-2** by performing a set of searches for "representative" C# exceptions. To identify this set of exceptions, we analyzed the 10 most popular projects on Codeplex[7]; Codeplex is a popular hosting site for Open Source C# projects. By analyzing the source code of these 10 projects, we identified the 18 most commonly-caught runtime exceptions. We decided to focus on runtime exceptions as we believe they are harder to resolve than compile-time errors. We split the experiment into three phases.

During the first phase, we performed a Dora search against each of the 18 exceptions. Each of the participants independently examined the first five results for each query to determine whether the result was relevant or not; we always looked at the top five results, regardless of Dora's thresholds (that would normally exclude poor results). The thresholds were ignored to increase the fairness to Google and Stack Overflow, both of which favour returning results over relevance. The participants determined relevance based on a result's ability to help solve the queried exception. Based on previous research that showed that developers only examine the first few results [22], we chose to investigate only the first five returned items.

During the second phase, the Google search was performed. Because Google web search results can depend on the past search history, we performed all our searches on one computer that was not logged into Google and had deleted all search cookies. We constructed the Google search query based on how we would perform the same search ourselves and thus created them by combining exception class names, parts of the exception message, and keywords that might describe the context. The results from these queries were saved, and each participant independently evaluated the relevance of the first five results. Here we considered all returned results (as a developer may need to in practice). The third phase involved performing the same steps as in the second phase but with the Stack Overflow website. We used the same queries that were created for Google search and independently evaluated the relevance of the returned results.

[7]http://codeplex.com

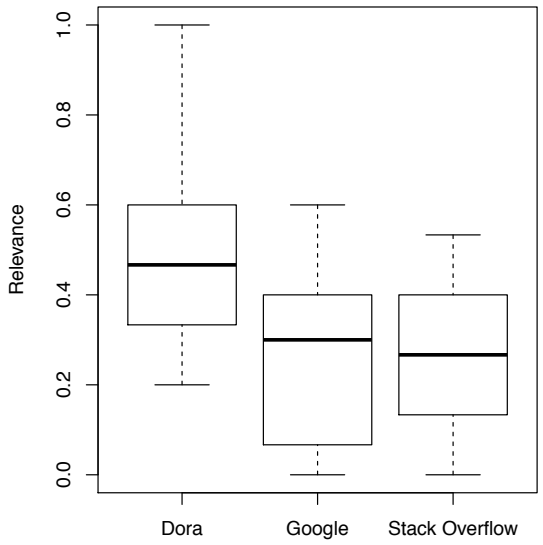

Fig. 4. Proportion of the first five results that are relevant. The box represents the first and third interquartile ranges; the heavy line represents the median. Higher values are better.

E. Results

We averaged the relevance results from each participant. Figure 4 demonstrates the differences in the proportion of relevant results for Dora, Google, and Stack Overflow among the first five. Dora only returned one result for one query; for all other queries five results were returned. The boxplot directly supports **RQ-1**: Dora improves the proportion of relevant results developers need to investigate for queries relating to specific programming errors. Dora provides 55% more relevant results than a Google web search and 75% more relevant results than a Stack Overflow search. Dora's median relevance (for the first five results) was 0.47 while Google's was 0.30 and Stack Overflow's was 0.27. Google seemed to perform better than Stack Overflow by returning results from multiple help forums (in addition to Stack Overflow).

In terms of the index of the first relevant result, the data also supports **RQ-2**: the first relevant Dora result was usually better than both Google and Stack Overflow (this is true for 14 of the 18 queries); these results are shown in Figure 5. The median index of the first relevant result for Dora was 1.0, while for Google it was 2.83 and for Stack Overflow it was 2.0. Interestingly, Stack Overflow performed better than Google; this may have been because the first two results from Google searches were often from the Microsoft MSDN Library and

MSDN Social Forums web sites which were deemed to be less relevant. Although MSDN Social Forums is an official place for community communication, it is not very popular and did not contain relevant information for any of the 18 queries.

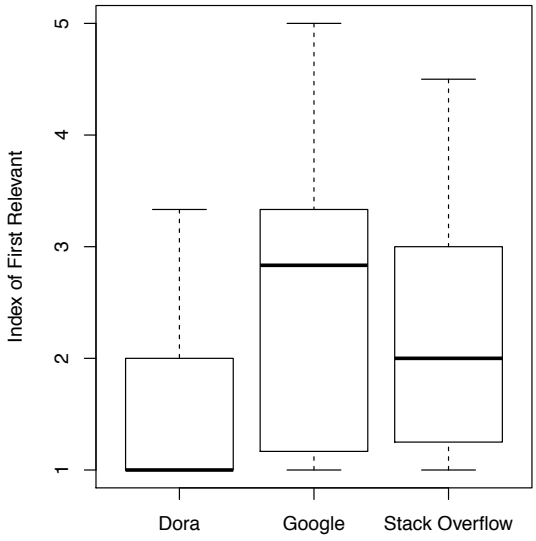

Fig. 5. Index of the first relevant result. The box represents the first and third interquartile ranges; the heavy line represents the median. Lower results are better.

We also evaluated the proportion of results in each of the three relevance categories. Dora returned at least one "good" result for 7 of the 18 queries; all 18 queries have a minimum of one "ok" result with an average value of 5.6 "ok" results. As shown in Figure 6, the proportion of results in each of the three categories is relatively constant between the queries from Section IV-B and Section V-C. Again, the results in the "poor" categories are not shown to the developer in practice.

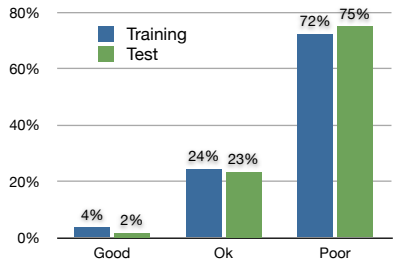

Fig. 6. Proportion of results found in each of the quality categories both data sets. Dora filters the result list to only display "good" and "ok" results to the developer.

F. Assessing the validity of the predictions based on the model

To get a sense of the accuracy of the model, we validated our pre-determined weights (set in Section IV-B) with an independent set of queries and relevance assessments. We revisited the 18 queries performed with Dora in Section V-D and evaluated all of the results Dora returned (instead of just the top 5); this produced 462 search results. The three participants individually went through the results and assigned values in the same way they did before. There were no major disagreements in the relevance assessment among the participants; the relevance assessments were averaged so each of the 462 results had a single relevance value.

By comparing the relevance value we applied to that generated by our model, the absolute mean error was 0.37. This means that the score returned by our model averaged only 0.37 away from our manually identified scores, where our scores were assigned a value between 1 and 5. The R^2 value, which measures goodness-of-fit, was 0.55. R^2 shows the proportion of variation among assigned values that is explained by the characteristics of our model. The correlation coefficient R that defines the overall correlation between values assigned by our model and by the participants was 0.74. From these results we conclude that although the model cannot predict the real values with 100% accuracy, the model's values and the manually assigned ones are connected by a statistically significant relationship.

VI. Discussion

In this section we discuss some threats to validity for our approach and evaluation, some limitations of our approach, and provide some recommendations for future work.

A. Threats to validity

In terms of internal validity, the personal experience of the experimental participants could have biased their selection of relevant results. Importantly, the determination of whether a result is relevant for a given task is a subjective measure. While the weighting we have applied works to identify results we deem relevant, it is possible that a different rubric for determining relevance would result in different model weights. We also only measured the relevance of a result. We did not further investigate how the knowledge gained from an irrelevant result could be used. For example, if developers learn more about their problem from examining irrelevant results, focusing solely on relevance may not be the right approach.

In terms of external validity, our prototype tool was geared towards queries targeting the C# language from within the Visual Studio IDE. Alternate languages, particularly rarer languages like TXL or Alloy, would likely require support for a different set of search sites. We also have only examined exceptional circumstances; our approach may not be applicable for higher-level knowledge (e.g., design knowledge).

B. Tool limitations and future work

The greatest limitation of our tool is that we require site-specific parsing rules to analyze a given site's posts to determine how they fit our model. This means that for general purpose queries, or for queries for which the programming language of the result is not important, our approach may not be general enough without querying a larger corpus of sites. In addition, our model may not translate to a non-discussion forum style of resource (i.e., blogs). For instance, we have identified the marked solved/answered characteristic

of a post to be the most useful in determining relevance. This characteristic does not translate well to blogs.

Further integration could enable new collaborative filtering possibilities (for instance, code being adopted from a specific response post could indicate an implicit acknowledgment of the value of the post). Also, further research could be conducted in transferring gained knowledge from online resources directly into the developer's system; this seems particularly tractable for compilation errors as the correct locality for the fix is well defined (e.g., where the error is identified by the compiler).

Finally, launching queries within the IDE enables us to generate a more contextually-relevant query (than a traditional keyword search). While we do not currently do this, a query could include logging or other feature-related information (for example, Query-Feature Graphs [23]). By experimenting with various levels of contextual information, Dora could start a search session with a very specific query and gradually relax the query once results are returned. In this way, Dora could automatically find the most descriptive query that returns results; this could potentially find results that are more contextually relevant than more general queries.

VII. Conclusion

Software developers frequently encounter compilation errors and runtime exceptions as they work on their systems. One common source of help for understanding these errors and exceptions is through archived developer conversations on the Internet. Unfortunately, finding relevant answers to specific programming problems online can be challenging due to the number of online resources that contain only questions or incorrect answers. We have built the Dora search tool to automatically identify helpful programming posts. Through a semi-controlled experiment, we found that Dora returned 55% more relevant results to developers than searching the web manually. We believe that Dora can help developers spend less time examining unhelpful online posts enabling them to concentrate more effectively on their core development tasks.

Acknowledgments

The authors would like to thank Olga Baysal for her assistance with WEKA and for providing valuable feedback on the work. We also thank the anonymous reviewers for their comments.

References

[1] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code," in *Proceedings of the Workshop on Evaluation and Usability of Programming Languages and Tools*, 2010, pp. 8:1–8:6.

[2] J. Stylos and B. A. Myers, "Mica: A web-search tool for finding API components and examples," in *Proceedings of the Visual Languages and Human-Centric Computing*, 2006, pp. 195–202.

[3] C. Spinuzzi, "Software development as mediated activity: Applying three analytical frameworks for studying compound mediation," in *Proceedings of the International Conference on Computer Documentation*, 2001, pp. 58–67.

[4] R. Hoffmann, J. Fogarty, and D. S. Weld, "Assieme: Finding and leveraging implicit references in a web search interface for programmers," in *Proceedings of the Symposium on User Interface Software and Technology*, 2007, pp. 13–22.

[5] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest Q&A site in the west," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2011, pp. 2857–2866.

[6] G. B. Duggan and S. J. Payne, "Knowledge in the head and on the web: Using topic expertise to aid search," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2008, pp. 39–48.

[7] B. V. Hanrahan, G. Convertino, and L. Nelson, "Modeling problem difficulty and expertise in StackOverflow," in *Proceedings of the conference on Computer Supported Cooperative Work*, 2012, pp. 91–94.

[8] R. McCauley, S. Fitzgerald, G. Lewandowski, L. Murphy, B. Simon, L. Thomas, and C. Zander, "Debugging: A review of the literature from an educational perspective," *Computer Science Education*, vol. 18, no. 2, pp. 67–92, 2008.

[9] M.-H. Nienaltowski, M. Pedroni, and B. Meyer, "Compiler error messages: What can help novices?" in *Proceedings of the Technical Symposium on Computer Science Education*, pp. 168–172.

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

[11] R. Holmes and R. J. Walker, "Supporting the investigation and planning of pragmatic reuse tasks," in *Proceedings of the International Conference on Software Engineering*, 2007, pp. 447–457.

[12] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 1589–1598.

[13] S. Henninger, "An evolutionary approach to constructing effective software reuse repositories," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 2, pp. 111–140, 1997.

[14] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid mining: Helping to navigate the API jungle," in *Proceedings of the Conference on Programming Language Design and Implementation*, 2005, pp. 48–61.

[15] R. Holmes and G. C. Murphy, "Using structural context to recommend source code examples," in *Proceedings of the International Conference on Software Engineering*, 2005, pp. 117–125.

[16] C. Parnin and C. Treude, "Measuring API documentation on the web," in *Proceedings of the International Workshop on Web 2.0 for Software Engineering*, 2011, pp. 25–30.

[17] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow," Georgia Institute of Technology, Tech. Rep.

[18] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing Stack Overflow for the IDE," in *Proceedings of the Workshop on Recommendation Systems for Software Engineering*, 2012.

[19] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and Trends of Information Retrieval*, vol. 2, no. 1-2, pp. 1–135, 2008.

[20] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[21] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proceedings of the International Conference on Software Engineering*, 2007, pp. 344–353.

[22] J. Starke, C. Luce, and J. Sillito, "Searching and skimming: An exploratory study," in *Proceedings of the International Conference on Software Maintenance*, 2009, pp. 157–166.

[23] A. Fourney, R. Mann, and M. Terry, "Query-feature graphs: Bridging user vocabulary and system functionality," in *Proceedings of the Symposium on User Interface Software and Technology*, 2011, pp. 207–216.