Exemplar: A Source Code Search Engine for Finding Highly Relevant Applications

Collin McMillan, *Member*, *IEEE*, Mark Grechanik, *Member*, *IEEE*, Denys Poshyvanyk, *Member*, *IEEE*, Chen Fu, *Member*, *IEEE*, and Qing Xie, *Member*, *IEEE*

Abstract—A fundamental problem of finding software applications that are highly relevant to development tasks is the mismatch between the high-level intent reflected in the descriptions of these tasks and low-level implementation details of applications. To reduce this mismatch we created an approach called *EXEcutable exaMPLes ARchive (Exemplar)* for finding highly relevant software projects from large archives of applications. After a programmer enters a natural-language query that contains high-level concepts (e.g., MIME, datasets), Exemplar retrieves applications that implement these concepts. Exemplar ranks applications in three ways. First, we consider the descriptions of applications. Second, we examine the Application Programming Interface (API) calls used by applications. Third, we analyze the dataflow among those API calls. We performed two case studies (with professional and student developers) to evaluate how these three rankings contribute to the quality of the search results from Exemplar. The results of our studies show that the combined ranking of application descriptions and API documents yields the most-relevant search results. We released Exemplar and our case study data to the public.

Index Terms—Source code search engines, information retrieval, concept location, open source software, mining software repositories, software reuse

1 Introduction

Programmers face many challenges when attempting to locate source code to reuse [42]. One key problem of finding relevant code is the mismatch between the high-level intent reflected in the descriptions of software and low-level implementation details. This problem is known as the *concept assignment problem* [6]. Search engines have been developed to address this problem by matching keywords in queries to words in the descriptions of applications, comments in their source code, and the names of program variables and types. These applications come from repositories which may contain thousands of software projects. Unfortunately, many repositories are polluted with poorly functioning projects [21]; a match between a keyword from the query with a word in the description or in the source code of an application does not guarantee that this application is relevant to the query.

Many source code search engines return snippets of code that are relevant to user queries. Programmers typically need to overcome a high cognitive distance [25] to understand how to use these code snippets. Moreover, many of these code fragments are likely to appear very similar [12]. If code fragments are retrieved in the contexts of executable applications, it makes it easier for programmers to understand how to reuse these code fragments.

Existing code search engines (e.g., Google Code Search, SourceForge (SF)) often treat code as plain text where all words have unknown semantics. However, applications contain functional abstractions in a form of API calls whose semantics are well defined. The idea of using API calls to improve code search was proposed and implemented elsewhere [14], [8]; however, it was not evaluated over a large codebase using a standard information retrieval methodology [30, pages 151-153].

We created an application search system called *EXEcutable exaMPLes ARchive (Exemplar)* as part of our *Searching, Selecting, and Synthesizing* (S^3) architecture [35]. Exemplar helps users find highly relevant executable applications for reuse. Exemplar combines three different sources of information about applications in order to locate relevant software: the textual descriptions of applications, the API calls used inside each application, and the dataflow among those API calls. We evaluated the contributions by these different types of information in two separate case studies. First, in Section 6, we compared Exemplar (in two configurations) to SourceForge. We analyzed the results of that study in Section 7 and created a new version of Exemplar. We evaluated our updates to Exemplar in Section 8. Our key finding is that our search engine's results improved when considering the API calls in applications instead of only the applications' descriptions. We have made Exemplar and the results of our case studies available to the public.[1]

2 Exemplar Approach

2.1 The Problem

A direct approach for finding highly relevant applications is to search through the descriptions and source code of applications to match keywords from queries to the names

- *C. McMillan and D. Poshyvanyk are with the Department of Computer Science, College of William & Mary, Williamsburg, VA 23185. E-mail: {cmc, denys}@cs.wm.edu.*
- *M. Grechanik, C. Fu, and Q. Xie are with Accenture Technology Labs, Chicago, IL 60601. E-mail: {mark.grechanik, chen.fu, qing.xie}@accenture.com.*

Manuscript received 21 Oct. 2010; revised 2 July 2011; accepted 14 July 2011; published online 10 Aug. 2011.
Recommended for acceptance by K. Inoue.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2010-10-0313.
Digital Object Identifier no. 10.1109/TSE.2011.84.

1. http://www.xemplar.org (verified 03/28/2011).

0098-5589/12/$31.00 © 2012 IEEE Published by the IEEE Computer Society

(a) Standard search engines.

(b) Exemplar search engine.

Fig. 1. Illustrations of the processes for standard and Exemplar search engines.

of program variables and types. This approach assumes that programmers choose meaningful names when creating source code, which is often not the case [2].

This problem is partially addressed by programmers who create meaningful descriptions of the applications in software repositories. However, state-of-the-art search engines use exact matches between the keywords from queries, the words in the descriptions, and the source code of applications. Unfortunately, it is difficult for users to guess exact keywords because "no single word can be chosen to describe a programming concept in the best way" [11]. The vocabulary chosen by a programmer is also related to the concept assignment problem because the terms in the high-level descriptions of applications may not match terms from the low-level implementation (e.g., identifier names and comments).

2.2 Key Ideas

Suppose that a programmer needs to encrypt and compress data. A programmer will naturally turn to a search engine such as SourceForge[2] and enter keywords such as `encrypt` and `compress`. The programmer then looks at the source code of the programs returned by these search engines to check to see if some API calls are used to encrypt and compress data. The presence of these API calls is a good starting point for deciding whether to check these applications further.

What we seek is to augment standard code search to include help documentations of widely used libraries, such as the standard *Java Development Kit (JDK)*.[3] Existing engines allow users to search for specific API calls, but knowing in advance what calls to search for is hard. Our idea is to match keywords from queries to words in help documentation for API calls. These help documents are descriptions of the functionality of API calls as well as the usage of those calls. In Exemplar, we extract the help documents that come in the form of *JavaDocs*. Programmers trust these documents because the documents come from known and respected vendors, were written by different people, were reviewed multiple times, and have been used by other programmers who report their experience at different forums [10].

We also observe that relations between concepts entered in queries are often reflected as dataflow links between API calls that implement these concepts in the program code. This observation is closely related to the concept of the *software reflexion models* formulated by Murphy, Notkin, and Sullivan. In these models, relations between elements of high-level models (e.g., processing elements of software architectures) are preserved in their implementations in source code [33], [32]. For example, if the user enters keywords `secure` and `send` and the corresponding API calls `encrypt` and `email` are connected via some dataflow, then an application with these connected API calls is more relevant to the query than applications where these calls are not connected.

Consider two API calls `string encrypt()` and `void email(string)`. After the call `encrypt` is invoked, it returns a string that is stored in some variable. At some later point a call to the function `email` is made and the variable is passed as the input parameter. In this case these functions are connected using a dataflow link which reflects the implicit logical connection between keywords in queries. Specifically, the data should be encrypted and then sent to some destination.

2.3 Motivating Example

Exemplar returns applications that implement the tasks described by the keywords in user queries. Consider the following task: Find an application for sharing, viewing, and exploring large datasets that are encoded using MIME, and the data can be stored using key value pairs. Using the following keywords, `MIME`, `type`, `data`, an unlikely candidate application called BIOLAP is retrieved using Exemplar with a high ranking score. The description of this application matches only the keyword `data`, and yet this application made it to the top 10 of the list.

BIOLAP uses the class `MimeType`, specifically its method `getParameterMap`, because it deals with MIME-encoded data. The descriptions of this class and this method contain the desired keywords, and these implementation details are highly relevant to the given task. BIOLAP does not show on the top 300 list of retrieved applications when the search is performed with the SourceForge search engine.

2.4 Fundamentals of Exemplar

Consider the process for standard search engines (e.g., Sourceforge, Google code search,[4] Krugle[5]) shown in Fig. 1a. A keyword from the query is matched against words in the descriptions of the applications in some repository (Sourceforge) or words in the entire corpus of source code (Google Code Search, Krugle). When a match is found, applications $\texttt{app}_1$ to $\texttt{app}_n$ are returned.

Consider the process for Exemplar shown in Fig. 1b. Keywords from the query are matched against the descriptions of different documents that describe API calls of widely used software packages. When a match is found, the names of the API calls $\texttt{call}_1$ to $\texttt{call}_k$ are returned. These names are matched against the names of the functions

2. http://sourceforge.net/ (verified 03/28/2011).

3. http://www.oracle.com/technetwork/java/javase/downloads/index.html (verified 03/28/2011).

4. http://www.google.com/codesearch (verified 03/28/2011).

5. http://opensearch.krugle.org (verified 03/28/2011).

invoked in these applications. When a match is found, applications $\texttt{app}_1$ to $\texttt{app}_n$ are returned.

In contrast to the keyword matching functionality of standard search engines, Exemplar matches keywords with the descriptions of the various API calls in help documents. Since a typical application invokes many API calls, the help documents associated with these API calls are usually written by different people who use different vocabularies. The richness of these vocabularies makes it more likely to find matches, and produce API calls $\texttt{APIcall}_1$ to $\texttt{APIcall}_k$. If some help document does not contain a desired match, some other document may yield a match. This is how we address the vocabulary problem [11].

As it is shown in Fig. 1b, API calls $\texttt{APIcall}_1$, $\texttt{APIcall}_2$, and $\texttt{APIcall}_3$ are invoked in the $\texttt{app}_1$. It is less probable that the search engine fails to find matches in help documents for all three API calls, and therefore the application $\texttt{app}_1$ will be retrieved from the repository.

Searching help documents produces additional benefits. API calls from help documents (that match query keywords) are linked to locations in the project source code where these API calls are used, thereby allowing programmers to navigate directly to these locations and see how high-level concepts from queries are implemented in the source code. Doing so solves an instance of the concept location problem [34].

3 Ranking Schemes

3.1 Components of Ranking

There are three components that compute different scores in the Exemplar ranking mechanism: a component that computes a score based on word occurrences in project descriptions (WOS), a component that computes a score based on the relevant API calls (RAS), and a component that computes a score based on dataflow connections between these calls (DCS). The total ranking score is the weighted sum of these three ranking scores.

We designed each ranking component to produce results from different perspectives (e.g., application descriptions, API calls, and dataflows among the API calls). The following three sections describe the components. Section 4 discusses the implementation of the components and includes important technical limitations that we considered when building Exemplar. We examine how WOS, RAS, and DCS each contribute to the results given by Exemplar in Section 7. Section 7 also covers the implications of our technical considerations.

3.2 WOS Ranking Scheme

The WOS component uses the *Vector Space Model* (VSM), which is a ranking function used by search engines to rank matching documents according to their relevance to a given search query. VSM is a bag-of-words retrieval technique that ranks a set of documents based on the terms appearing in each document as well as the query. Each document is modeled as a vector of the terms it contains. The weights of those terms in each document are calculated in accordance to the *Term Frequency/Inverse Document Frequency* (TF/IDF). Using TF/IDF, the weight for a term is calculated as $tf = \frac{n}{\sum_k n_k}$, where n is the number of occurrences of the term in the document, and $\sum_k n_k$ is the sum of the number of occurrences of the term in all documents. Then, the similarities among the documents are calculated using the cosine distance between each pair of documents $cos(\theta) = \frac{d_1 \cdot d_2}{\|d_1\|\|d_2\|}$, where d_1 and d_2 are document vectors.

3.3 RAS Ranking Scheme

The documents in our approach are the different documents that describe each API call (e.g., each JavaDoc). The collection of API documents is defined as $D_{API} = (D^1_{API}, D^2_{API}, \ldots, D^k_{API})$. A corpus is created from D_{API} and represented as the term-by-document $m \times k$ matrix M, where m is the number of terms and k is the number of API documents in the collection. A generic entry $a[i, j]$ in this matrix denotes a measure of the weight of the ith term in the jth API document [40].

API calls that are relevant to the user query are obtained by ranking documents, D_{API}, that describe these calls as relevant to the query Q. This relevance is computed as a conceptual similarity, C, (i.e., the length-normalized inner product) between the user query, Q, and each API document, D_{API}. As a result the set of triples $\langle A, C, n\rangle$ is returned, where A is the API call, n is the number of occurrences of this API call in the application with the conceptual similarity, C, of the API call documentation to query terms.

The API call-based ranking score for the application, j, is computed as

$$S^j_{ras} = \frac{\sum_{i=1}^{p} n^j_i \cdot C^j_i}{|A|^j},$$

where $|A|^j$ is the total number of API calls in the application j and p is the number of API calls retrieved for the query.

3.4 DCS Ranking Scheme

To improve the precision of ranking we derive the structure of connections between API calls and use this structure as an important component in computing rankings. The standard syntax for invoking an API call is $\texttt{t var} = \texttt{o.callname}(p_1, \ldots, p_n)$. The structural relations between API calls reflect compositional properties between these calls. Specifically, it means that API calls access and manipulate data at the same memory locations.

There are four types of dependencies between API calls: input, output, true, and antidependence [31, page 268]. True dependence occurs when the API call `f` write a memory location that the API call `g` later reads (e.g., $\texttt{var} = \texttt{f}(\ldots); \ldots; \texttt{g}(\texttt{var}, \ldots);$). Antidependence occurs when the API call `f` reads a memory location that the API call `g` later writes (e.g., $\texttt{f}(\texttt{var}, \ldots), \ldots; \texttt{var} = \texttt{g}(\ldots);$). Output dependence occurs when the API calls `f` and `g` write the same memory location. Finally, input dependence occurs when the API calls `f` and `g` read the same memory location.

Consider an all-connected graph (i.e., a clique) where nodes are API calls and the edges represent dependencies among these calls for one application. The absence of an edge means that there is no dependency between two API calls. Let the total number of connections among n retrieved API calls be less than or equal to $n(n-1)$. Let a connection between two distinct API calls in the application be defined as *Link*; we assign some weight w to this Link based on the

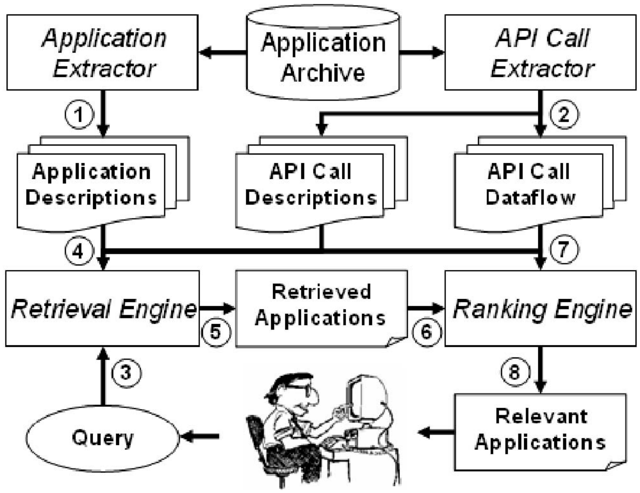


Fig. 2. Exemplar architecture.

strength of the dataflow or control flow dependency type. The ranking is normalized to be between 0 and 1.

The API call connectivity-based ranking score for the application, j, is computed as

$$S_{dcs}^{j} = \frac{\sum_{i=1}^{n(n-1)} w_i^j}{n(n-1)},$$

where w_i is the weight to each type of flow dependency for the given link $Link$ such that $1 > w_i^{true} > w_i^{anti} > w_i^{output} > w_i^{input} > 0$. The intuition behind using this order is that these dependencies contribute differently to ranking heuristics. Specifically, using the values of the same variable in two API calls introduces a weaker link as compared to the true dependency where one API call produces a value that is used in some other API call.

3.5 Integrated Scheme

The final ranking score is computed as $S = \lambda_{wos} S_{wos} + \lambda_{ras} S_{ras} + \lambda_{dcs} S_{dcs}$, where λ is the interpolation weight for each type of the score. These weights are determined independently of queries, unlike the scores, which are query dependent. Adjusting these weights enables experimentation with how underlying structural and textual information in application affects resulting ranking scores. The formula for S remains the same throughout this paper, and all three weights were equal during the case study in Section 5. We explore alterations to Exemplar, including λ, based on the case study results in Section 7.

4 Implementation Details

Fig. 2 shows the architecture of Exemplar. In this section, we step through Fig. 2 and describe some technical details behind Exemplar.

Two crawlers, *Application Extractor* and *API Call Extractor*, populate Exemplar with data from SourceForge. We currently have run the crawlers on SourceForge and obtained more than 8,000 Java projects containing 414,357 files.[6] The Application Extractor downloads the applications and extracts the descriptions and source code of those applications (the Application Metadata (1)). The API Call Extractor crawls the source code from the applications for the API calls that they use, the descriptions of the API calls, and the dataflow among those calls (the API Call Metadata (2)). The API Call Extractor ran with 65 threads for over 50 hours on 30 computers: Three machines have two dual-core 3.8 Ghz EM64T Xeon processors with 8 Gb RAM, two have four 3.0 Ghz EM64T Xeon CPUs with 32 Gb RAM, and the rest have one 2.83 Ghz quad-core CPU and 2 Gb RAM. The API Call Extractor found nearly 12 million API invocations from the JDK 1.5 in the applications. It also processes the API calls for their descriptions, which in our case are the JavaDocs for those API calls.

Our approach relies on the tool PMD[7] for computing approximate dataflow links, which are based on the patterns described in Section 3.4. PMD extracts data from individual Java source files, so we are only able to locate dataflow links among the API calls as they are used in any one file. We follow the variables visible in each scope (e.g., global variables plus those declared in methods). We then look at each API call in the scope of those variables. We collect the input parameters and output of those API calls. We then analyze this input and output for dataflow. For example, if the output of one API call is stored in a variable which is then used as input to another API call, then there is dataflow between those API calls. Note that our technique is an approximation and can produce both false positive and false negatives. Determining the effects of this approximation on the quality of Exemplar's results is an area of future work.

The *Retrieval Engine* locates applications in two ways (3). First, the input to the Retrieval Engine is the user query, and the engine matches keywords in this query (5) to keywords in the descriptions of applications. Second, the Retrieval Engine finds descriptions of API calls which match keywords.[8] The Retrieval Engine then locates applications which use those API calls. The engine outputs a list of Retrieved Applications (6).

The *Ranking Engine* uses the three ranking schemes from Section 3 (WOS, RAS, and DCS) to sort the list of retrieved applications (7). The Ranking Engine depends on three sources of information: descriptions of applications, the API calls used by each application, and the dataflow among those API calls (4). The Ranking Engine uses Lucene,[9] which is based on VSM, to implement WOS. The combination of the ranking schemes (see Section 3.5) determines the relevancy of the applications. The Relevant Applications are then presented to the user (8).

5 Case Study Design

Typically, search engines are evaluated using manual relevance judgments by experts [30, pages 151-153]. To determine how effective Exemplar is, we conducted a case study with 39 participants who are professional programmers. We gave a list of tasks described in English. Our goal is to evaluate how well these participants can find applications

6. We ran the crawlers in August 2009.

7. http://pmd.sourceforge.net/ (verified 03/28/2011).

8. Exemplar limits the number of relevant API calls it retrieves for each query to 200. This limit was necessary due to performance constraints. See Section 7.4.

9. http://lucene.apache.org (verified 03/28/2011).

TABLE 1
Plan for the Case Study of Exemplar and Sourceforge

Experiment	Group	Search Engine	Task Set
1	G1	EWD	T1
	G2	SF	T2
	G3	END	T3
2	G1	END	T2
	G2	EWD	T3
	G3	SF	T1
3	G1	SF	T3
	G2	END	T1
	G3	EWD	T2

that match given tasks using three different search engines: Sourceforge (SF) and Exemplar with (EWD) and without (END) dataflow links as part of the ranking mechanism. We chose to compare Exemplar with Sourceforge because the latter has a popular search engine with the largest open source Java project repository, and Exemplar is populated with Java projects from this repository.

5.1 Methodology

We used a cross-validation study design in a cohort of 39 participants who were randomly divided into three groups. We performed three separate experiments during the study. In each experiment, each group was given a different search engine (i.e., SF, EWD, or END), as shown in Table 1. Then, in the experiments each group would be asked to use a different search engine than that group had used before. The participants would use the assigned engine to find applications for given tasks. Each group used a different set of tasks in each experiment. Thus, each participant used each search engine on different tasks in this case study. Before the study we gave a one-hour tutorial on using these search engines to find applications for tasks.

Each experiment consisted of three steps. First, participants translated tasks into a sequence of keywords that described key concepts of applications that they needed to find. Then, participants entered these keywords as queries into the search engines (the order of these keywords does not matter) and obtained lists of applications that were ranked in descending order.

The next step was to examine the returned applications and to determine if they matched the tasks. Each participant accomplished this step by him or herself, assigning a confidence level, C, to the examined applications using a four-level Likert scale. We asked participants to examine only the top 10 applications that resulted from their searches. We evaluated only the top 10 results because users of search engines rarely look beyond the 10th result [13] and because other source code search engines have been evaluated using the same number of results [19].

The guidelines for assigning confidence levels are the following:

1. Completely irrelevant—there is absolutely nothing that the participant can use from this retrieved project, nothing in it is related to your keywords.
2. Mostly irrelevant—only a few remotely relevant code snippets or API calls are located in the project.
3. Mostly relevant—a somewhat large number of relevant code snippets or API calls in the project.
4. Highly relevant—the participant is confident that code snippets or API calls in the project can be reused.

Twenty-six participants were Accenture employees who work on consulting engagements as professional Java programmers for different client companies. The remaining 13 participants were graduate students from the University of Illinois at Chicago who have at least six months of Java experience. The Accenture participants had different backgrounds, experience, and belonged to different groups of the total Accenture workforce of approximately 180,000 employees. Out of 39 participants, 17 had programming experience with Java ranging from one to three years, and 22 participants reported more than three years of experience writing programs in Java. Eleven participants reported prior experience with Sourceforge (which is used in this case study), 18 participants reported prior experience with other search engines, and 11 said that they never used code search engines. Twenty-six participants had bachelor degrees and 13 had master's degrees in different technical disciplines.

5.2 Precision

The two main measures for evaluating the effectiveness of retrieval are precision and recall [49, page 188-191]. The precision is calculated as $P_r = \frac{\text{relevant}}{\text{retrieved}}$, where `relevant` is the number of retrieved applications that are relevant and `retrieved` is the total number of applications retrieved. The precision of a ranking method is the fraction of the top r ranked documents that are relevant to the query, where $r = 10$ in this case study. Relevant applications are counted only if they are ranked with the confidence levels `4` or `3`. The precision metrics reflects the accuracy of the search. Since we limit the investigation of the retrieved applications to top 10, the recall is not measured in this study.

5.3 Discounted Cumulative Gain (DCG)

Discounted Cumulative Gain is a metric for analyzing the effectiveness of search engine results [1]. The intuition behind DCG is that search engines should not only return relevant results, but should rank those results by relevancy. Therefore, DCG rewards search engines for ranking relevant results above irrelevant ones. We calculate the DCG for the top 10 results from each engine because we collect confidence values for these results. We compute DCG according to this formula: $G = C_1 + \sum_{i=2}^{10} \frac{C_i}{log_2\, i}$, where C_1 is the confidence value of the result in the first position and C_i is the confidence value of the result in the ith position. We normalize the DCG using the following formula: $NG = \frac{G}{iG}$, where iG is the ideal DCG in the case when the confidence value for the first 10 results is always 4 (indicating that all 10 results are highly relevant). We refer to normalized DCG as NG in the remainder of this paper.

5.4 Hypotheses

We introduce the following null and alternative hypotheses to evaluate how close the means are for the confidence levels (Cs) and precisions (Ps) for control and treatment groups. Unless we specify otherwise, participants of the treatment group use either END or EWD, and participants of the control group use SF. We seek to evaluate the following hypotheses at a 0.05 level of significance:

- $H_{0\text{-}null}$: The primary null hypothesis is that there is no difference in the values of confidence level and precision per task between participants who use SF, EWD, and END.
- $H_{0\text{-}alt}$: An alternative hypothesis to H_{0-null} is that there is a statistically significant difference in the values of confidence level and precision between participants who use SF, EWD, and END.

Once we test the null hypothesis H_{0-null}, we are interested in the directionality of means, μ, of the results of control and treatment groups. We are interested to compare the effectiveness of EWD versus the END and SF with respect to the values of C, P, and NG.

- H_1 (C of EWD versus SF): The effective null hypothesis is that $\mu_C^{EWD} = \mu_C^{SF}$, while the true null hypothesis is that $\mu_C^{EWD} \leq \mu_C^{SF}$. Conversely, the alternative hypothesis is $\mu_C^{EWD} > \mu_C^{SF}$.
- H_2 (P of EWD versus SF): The effective null hypothesis is that $\mu_P^{EWD} = \mu_P^{SF}$, while the true null hypothesis is that $\mu_P^{EWD} \leq \mu_P^{SF}$. Conversely, the alternative hypothesis is $\mu_P^{EWD} > \mu_P^{SF}$.
- H_3 (NG of EWD versus SF): The effective null hypothesis is that $\mu_{NG}^{EWD} = \mu_{NG}^{SF}$, while the true null hypothesis is that $\mu_{NG}^{EWD} \leq \mu_{NG}^{SF}$. Conversely, the alternative hypothesis is $\mu_{NG}^{EWD} > \mu_{NG}^{SF}$.
- H_4 (C of EWD versus END): The effective null hypothesis is that $\mu_C^{EWD} = \mu_C^{END}$, while the true null hypothesis is that $\mu_C^{EWD} \leq \mu_C^{END}$. Conversely, the alternative is $\mu_C^{EWD} > \mu_C^{END}$.
- H_5 (P of EWD versus END): The effective null hypothesis is that $\mu_P^{EWD} = \mu_P^{END}$, while the true null hypothesis is that $\mu_P^{EWD} \leq \mu_P^{END}$. Conversely, the alternative is $\mu_P^{EWD} > \mu_P^{END}$.
- H_6 (NG of EWD versus END): The effective null hypothesis is that $\mu_{NG}^{EWD} = \mu_{NG}^{END}$, while the true null hypothesis is that $\mu_{NG}^{EWD} \leq \mu_{NG}^{END}$. Conversely, the alternative is $\mu_{NG}^{EWD} > \mu_{NG}^{END}$.
- H_7 (C of END versus SF): The effective null hypothesis is that $\mu_C^{END} = \mu_C^{SF}$, while the true null hypothesis is that $\mu_C^{END} \leq \mu_C^{SF}$. Conversely, the alternative hypothesis is $\mu_C^{END} > \mu_C^{SF}$.
- H_8 (P of END versus SF): The effective null hypothesis is that $\mu_P^{END} = \mu_P^{SF}$, while the true null hypothesis is that $\mu_P^{END} \leq \mu_P^{SF}$. Conversely, the alternative hypothesis is $\mu_P^{END} > \mu_P^{SF}$.
- H_9 (NG of END versus SF): The effective null hypothesis is that $\mu_{NG}^{END} = \mu_{NG}^{SF}$, while the true null hypothesis is that $\mu_{NG}^{END} \leq \mu_{NG}^{SF}$. Conversely, the alternative hypothesis is $\mu_{NG}^{END} > \mu_{NG}^{SF}$.

The rationale behind the alternative hypotheses to H_1, H_2, and H_3 is that Exemplar allows users to quickly understand how keywords in queries are related to implementations using API calls in retrieved applications. The alternative hypotheses to H_4, H_5, H_6 are motivated by the fact that if users see dataflow connections between API calls, they can make better decisions about how closely retrieved applications match given tasks. Finally, having the alternative hypotheses to H_7, H_8, and H_9 ensures that Exemplar without dataflow links still allows users to quickly understand how keywords in queries are related to implementations using API calls in retrieved applications.

5.5 Task Design

We designed 26 tasks that participants work on during experiments in a way that these tasks belong to domains that are easy to understand, and they have similar complexity. The following are two example tasks; all others may be downloaded from the Exemplar about page.[10]

1. "Develop a universal sound and voice system that allows users to talk, record audio, and play MIDI records. Users should be able to use open source connections with each other and communicate. A GUI should enable users to save conversations and replay sounds."
2. "Implement an application that performs pattern matching operations on a character sequences in the input text files. The application should support iterating through the found sequences that match the pattern. In addition, the application should support replacing every subsequence of the input sequence that matches the pattern with the given replacement string."

Additional criteria for these tasks is that they should represent real-world programming tasks and should not be biased toward any of the search engines that are used in this experiment. Descriptions of these tasks should be flexible enough to allow participants to suggest different keywords for searching. This criteria significantly reduces any bias toward evaluated search engines.

5.6 Normalizing Sources of Variations

Sources of variation are all issues that could cause an observation to have a different value from another observation. We identify sources of variation as the prior experience of the participants with specific applications retrieved by the search engines in this study, the amount of time they spend on learning how to use search engines, and different computing environments which they use to evaluate retrieved applications. The first point is sensitive since some participants who already know how some retrieved applications behave are likely to be much more effective than other participants who know nothing of these applications.

We design this experiment to drastically reduce the effects of covariates (i.e., nuisance factors) in order to normalize sources of variations. Using the cross-validation design, we normalize variations to a certain degree since each participant uses all three search engines on different tasks.

5.7 Tests and the Normality Assumption

We use one-way ANOVA and randomization tests [44] to evaluate the hypotheses. ANOVA is based on an assumption that the population is normally distributed. The law of large numbers states that if the population sample is sufficiently large (between 30 to 50 participants), then the central limit theorem applies even if the population is not

10. http://www.cs.wm.edu/semeru/exemplar/#casestudy (verified 03/28/2011).

normally distributed [43, pages 244-245]. Since we have 39 participants, the central limit theorem applies, and the above-mentioned tests have statistical significance.

5.8 Threats to Validity

In this section, we discuss threats to the validity of this case study and how we address these threats.

5.8.1 Internal Validity

Internal validity refers to the degree of validity of statements about cause-effect inferences. In the context of our experiment, threats to internal validity come from confounding the effects of differences among participants, tasks, and time pressure.

Participants. Since evaluating hypotheses is based on the data collected from participants, we identify two threats to internal validity: Java proficiency and motivation of participants.

Even though we selected participants who have working knowledge of Java as it was documented by human resources, we did not conduct an independent assessment of how proficient these participants are in Java. The danger of having poor Java programmers as participants of our case study is that they can make poor choices of which retrieved applications better match their queries. This threat is mitigated by the fact that all participants from Accenture worked on successful commercial projects as Java programmers.

The other threat to validity is that not all participants could be motivated sufficiently to evaluate retrieved applications. We addressed this threat by asking participants to explain in a couple of sentences why they chose to assign a certain confidence level to applications, and based on their results we financially awarded top five performers.

Tasks. Improper tasks pose a big threat to validity. If tasks are too general or trivial (e.g., open a file and read its data into memory), then every application that has file-related API calls will be retrieved, thus creating bias toward Exemplar. On the other hand, if application and domain-specific keywords describe the task (e.g., `genealogy` and `GENTECH`), only a few applications will be retrieved whose descriptions contain these keywords, thus creating a bias toward Sourceforge. To avoid this threat, we based the task descriptions on a dozen specifications of different software systems that were written by different people for different companies. The tasks we used in the case study are available for download at the Exemplar website.[11]

Time pressure. Each experiment lasted for 2 hours, and for some participants it was not enough time to explore all retrieved applications for each of eight tasks. It is a threat to validity that some participants could try to accomplish more tasks by shallowly evaluating retrieved applications. To counter this threat we notified participants that their results would be discarded if we did not see sufficient reported evidence of why they evaluated retrieved applications with certain confidence levels.

5.8.2 External Validity

To make the results of this case study generalizable, we must address threats to external validity, which refer to the generalizability of a casual relationship beyond the circumstances of our case study. The fact that supports the validity of the case study design is that the participants are highly representative of professional Java programmers. However, a threat to external validity concerns the usage of search tools in the industrial settings, where requirements are updated on a regular basis. Programmers use these updated requirements to refine their queries and locate relevant applications using multiple iterations of working with search engines. We addressed this threat only partially by allowing programmers to refine their queries multiple times.

In addition, it is sometimes the case that when engineers perform multiple searches using different combinations of keywords, they select certain retrieved applications from each of these search results. We believe that the results produced by asking participants to decide on keywords and then perform a single search and rank applications do not deviate significantly from the situation where searches using multiple (refined) queries are performed.

Another threat to external validity comes from different sizes of software repositories. We populated Exemplar's repository with all Java projects from the Sourceforge repository to address this threat to external validity.

Finally, the help documentation that we index in Exemplar is an external threat to validity because this documentation is provided by a third party, and its content and format may vary. We addressed this threat to validity by using the Java documentation extracted as JavaDocs from the official Java Development Kit, which has a uniform format.

6 Empirical Results

In this section, we report the results of the case study and evaluate the null hypotheses.

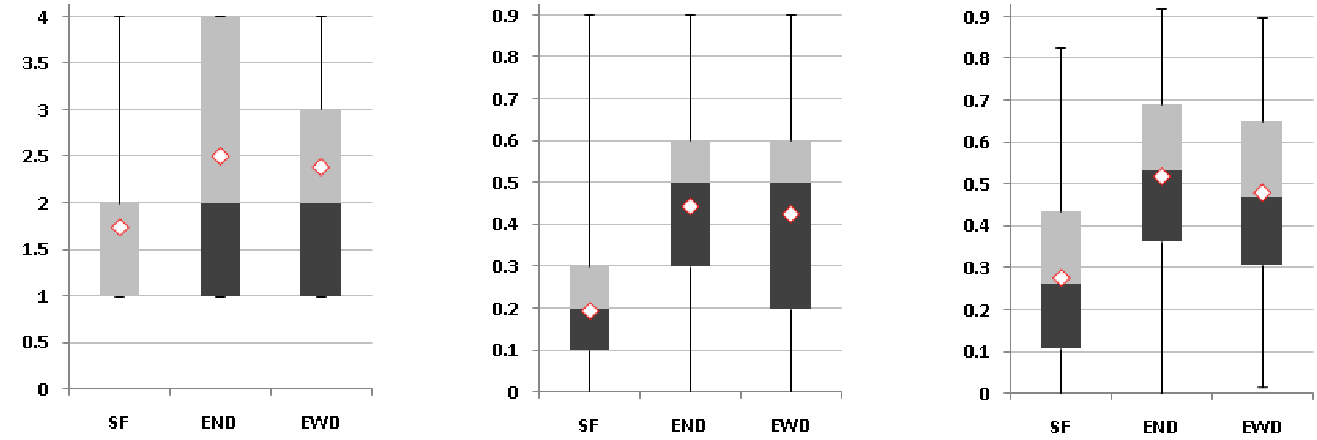
6.1 Variables

A main independent variable is the search engine (SF, EWD, END) that participants use to find relevant Java applications. Dependent variables are the values of confidence level, C, precision, P, and normalized discounted cumulative gain, NG. We report these variables in this section. The effect of other variables (task description length, prior knowledge) is minimized by the design of this case study.

6.2 Testing the Null Hypothesis

We used ANOVA [43] to evaluate the null hypothesis H_{0-null} that the variation in an experiment is no greater than that due to normal variation of individuals' characteristics and error in their measurement. The results of ANOVA confirm that there are large differences between the groups for C with $F = 129 > F_{crit} = 3$ with $p \approx 6.4 \cdot 10^{-55}$, which is strongly statistically significant. The mean C for the SF approach is 1.83 with the variance 1.02, which is smaller than the mean C for END, 2.47 with the variance 1.27, and it is smaller than the mean C for EWD, 2.35 with the variance 1.19. Also, the results of ANOVA confirm that there are large differences between the groups for P with $F = 14 > F_{crit} = 3.1$ with $p \approx 4 \cdot 10^{-6}$, which is strongly statistically significant. The mean P for the SF approach is 0.27 with the variance 0.03, which is smaller than the mean P for END, 0.47 with the variance 0.03, and it is smaller than the mean

11. http://www.xemplar.org , follow the "About Exemplar" link to the "Case Study" section.



(a) Confidence level, C. (b) Precision, P. (c) Normalized Discounted Gain, NG.

Fig. 3. Statistical summary of the results of the case study for C and P. The center point represents the mean. The dark and light gray boxes are the lower and upper quartiles, respectively. The thin line extends from the minimum to the maximum value.

P for EWD, 0.41 with the variance 0.026. Based on these results we reject the null hypothesis and we accept the alternative hypothesis H_{0-alt}.

A statistical summary of the results of the case study for C, P, and NG (median, quartiles, range, and extreme values) is shown as box-and-whisker plots in Figs. 3a, 3b, and 3c, correspondingly, with 95 percent confidence interval for the mean.

6.3 Comparing Sourceforge with Exemplar

To test the null hypothesis H_1, H_2, H_3, H_7, H_8, and H_9 we applied six randomization tests for C, P, and NG for participants who used SF and both variants of Exemplar. The results of this test are shown in Table 2. The column `Samples` shows that 37 out of a total of 39 participants participated in all experiments and created rankings for P (two participants missed one experiment). `Samples` indicates the number of results which were ranked in the case of variable C. For NG, `Samples` shows the number of sets of results. Based on these results we reject the null hypotheses H_1, H_2, H_3, H_7, H_8, and H_9, and we accept the alternative hypotheses that states that **participants who use Exemplar report higher relevance and precision on finding relevant applications than those who use Sourceforge**.

TABLE 2
Results of Randomization Tests of Hypotheses, H, for Dependent Variable Specified in the Column Var (C, P, or NG) Whose Measurements Are Reported in the Following Columns

H	Var	Approach	Samples	Min	Max	Median	μ	C	p
H_1	C	EWD	1273	1	4	2	2.35	-0.02	< 0.0001
		SF	1273	1	4	1	1.82		
H_2	P	EWD	76	0.12	0.74	0.42	0.41	0.34	< 0.0001
		SF	76	0.075	0.73	0.48	0.46		
H_3	NG	EWD	76	0.02	0.89	0.47	0.48	-0.05	< 0.0001
		SF	76	0	0.83	0.26	0.28		
H_4	C	EWD	1273	1	4	2	2.35	0.01	< 0.0001
		END	1273	1	4	3	2.47		
H_5	P	EWD	76	0.12	0.74	0.42	0.41	0.41	0.78927
		END	76	0.075	0.73	0.48	0.46		
H_6	NG	EWD	76	0.02	0.89	0.47	0.48	-0.02	0.71256
		END	76	0	0.92	0.53	0.52		
H_7	C	END	1307	1	4	3	2.47	-0.02	< 0.0001
		SF	1307	1	4	1	1.84		
H_8	P	END	76	0.075	0.73	0.5	0.47	0.4	< 0.0001
		SF	76	0	0.71	0.24	0.27		
H_9	NG	END	76	0	0.92	0.53	0.52	0.08	< 0.0001
		SF	76	0	0.83	0.26	0.28		

Extremal values, median, means, μ, and the Pearson correlation coefficient, C, are reported along with the results of the evaluation of the hypotheses, i.e., statistical significance, p.

6.4 Comparing EWD with END

To test the null hypotheses H_4, H_5, and H_6, we applied two t-tests for paired two sample for means, for C, P, and NG for participants who used END and EWD. The results of this test are shown in Table 2. Based on these results we reject the null hypothesis H_4, and say that **participants who use END report higher relevance when finding relevant applications than those who use EWD**. On the other hand, we fail to accept the null hypotheses H_5 and H_6, and say that **participants who use END do not report higher precision or normalized discounted cumulative gain than those who use EWD**.

There are several explanations for this result. First, given that our dataflow analysis is imperfect, some links are missed and, subsequently, the remaining links cannot affect the ranking score significantly. Second, it is possible that our dataflow connectivity-based ranking mechanism needs fine-tuning, and it is a subject of our future work. Finally, after the case study, a few participants questioned the idea of dataflow connections between API calls. A few participants had vague ideas as to what dataflow connections meant and how to incorporate them into the evaluation process. This phenomenon points to a need for better descriptions of Exemplar's internals in any future case studies.

6.5 Qualitative Analysis and User Comments

Thirty-five of the participants in the case study completed exit surveys (see Table 3) describing their experiences and opinions. Of these, 22 reported that seeing standalone fragments of the code alongside relevant applications would be more useful than seeing only software applications. Only four preferred simply applications listed in the results, while nine felt that either would be useful. Several users stated that seeing entire relevant applications provides useful context for code fragments, while others read code in order to understand certain algorithms or processes, but ultimately reimplement the functionality themselves.

TABLE 3
The Seven Questions Answered by the Case Study Participants during the Exit Survey

	Question
1	How many years of programming experience do you have?
2	What programming languages have you used and for how many years each?
3	How often do you use code search engines?
4	What code search engines have you used and for how long?
5	How often can you reuse found applications or code fragments in your work?
6	What is the biggest impediment to using code search engines, in your opinion?
7	Would you rather be able to retrieve a standalone fragment of code or an entire application with a relevant fragment of code in it?

All questions were open-ended.

After performing the case study, we responded to these comments by providing the source code directly on Exemplars results page, with links to the lines of files where relevant API calls are used. This constitutes a new feature of Exemplar which was not available to the participants during the user study.

Nineteen of the participants reported using source code search engines rarely, six said they sometimes use source code search engines, and nine regularly. Of those that only rarely use source code search engines, eight adapted Google's web search to look for code. Meanwhile, when asked to state the biggest impediment in using source code search engines, 14 participants answered that existing engines return irrelevant results, four were mostly concerned with the quality of the returned source code, six did not answer, and 11 reported some other impediment. These results support the recent studies [42] and point to a strong need for improved code engines that return focused, relevant results. New engines should show the specific processes and useful fragments of code. We believe that searching by API calls can fill this role because calls have specific and well-defined semantics along with high-quality documentation.

The following is a selection of comments written by participants in the user study. Scanned copies of all questionnaires are publicly available on the Exemplar about page.

- "The Exemplar search is handy for finding the APIs quickly."
- "Many SourceForge projects [have] no files or archives."
- "A stand-alone fragment would be easy to see and determine relevance to my needs, but an entire application would allow for viewing context which would be useful."
- "[I] typically reuse the pattern/algorithm, not [the] full code."
- "Often [retrieved code or applications] give me a clue as to how to approach a development task, but usually the code is too specific to reuse without many changes."
- "Often, [with source code search engines] I find results that do not have code."
- "[I reuse code] not in its entirety, but [I] always find inspiration."
- "There seems to be a lot of time needed to understand the code found before it can be usefully applied."
- "Could the line number reference [in Exemplar] invoke a collapsible look at the code snippet?"
- "With proper keywords used, [Exemplar] is very impressive. However, it does not filter well the executables and noncode files. Overall, great for retrieving simple code snippets."
- "Most, if not all, results returned [by Exemplar] provided valuable direction/foundation for completing the required tasks."
- "During this experiment it became clear that searching for API can be much more effective than by keywords in many instances. This is because it is the APIs that determine functionality and scope potential."
- "SourceForge was not as easy to find relevant software as hoped for."
- "[Using SourceForge] I definitely missed the report within Exemplar that displays the matching API methods/calls."
- "SourceForge appears to be fairly unreliable for projects to actually contain any files."
- "Exemplar seems much more intuitive and easier to use than SourceForge."
- "Great tool to find APIs through projects."
- "It was really helpful to know what API calls have been implemented in the project while using Exemplar."

The users were overall satisfied with Exemplar, preferring it to SourceForge's search. In Section 6, we found that they rated results from Exemplar with statistically significantly higher confidence levels than SourceForge. From our examination of these surveys, we confirm the findings from our analysis in Section 6 and conclude that the participants in the case study did prefer to search for applications using Exemplar rather than SourceForge. Moreover, we conclude that the reason they preferred Exemplar is because of Exemplar's search of API documentation.

7 Analysis of User Study Results

During our case study of Exemplar (see Section 5), we found that the original version of Exemplar outperformed SourceForge in terms of both confidence and precision. In this section, we will explore why Exemplar outperformed SourceForge. Our goal is to identify which components of Exemplar led to the improvements and to determine how users interpreted tasks and interacted with the source code search engine. Specifically, we intend to answer the following research questions (RQ):

- RQ_1. Do high Exemplar scores actually match high confidence level ranks from the participants?
- RQ_2. Do the components of the Exemplar score (WOS, RAS, and DCS scores) indicate relevance of applications when the others do not (e.g., do the

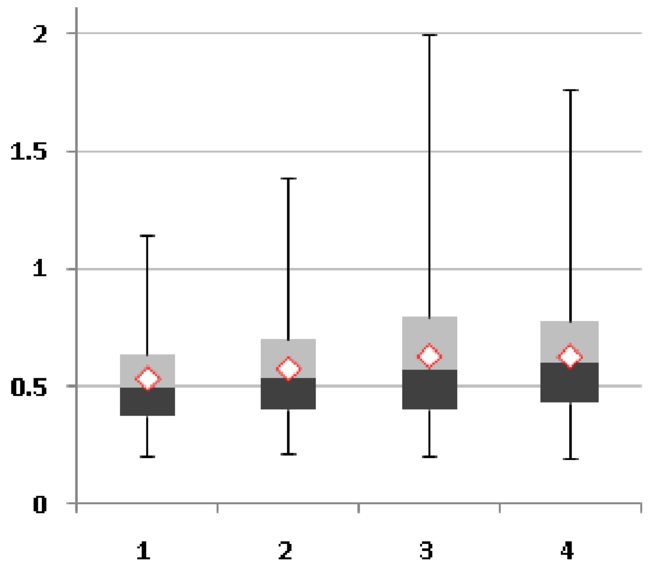


Fig. 4. Statistical summary of the scores from the case study of Exemplar. The y-axis is the score given by Exemplar during the case study. The x-axis is the confidence level given by users to results from Exemplar.

components capture the same or orthogonal information about retrieved software applications)?

- RQ_3. Is Exemplar sensitive to differences in the user queries when those queries were generated for the same task by different users?

We want to know how we can optimize Exemplar given answers to these research questions. Additionally, we want to study how design decisions (such as whether RAS considers the frequency of API calls, see Section 4) affected Exemplar.

7.1 Comparing Scores in Confidence Levels

Exemplar computes a score for every application to represent that application's relevance to the user query (see Section 4). Ideally, higher scores will be attached to applications with greater relevance. We know from Section 6 that Exemplar returns many relevant results, but this information alone is insufficient to claim that a high score from Exemplar for an application is actually an indicator of the relevance of that application because irrelevant applications could still obtain high scores (see Section 9).

To better understand the relationship of Exemplar ranking scores to relevance of retrieved software applications and to answer RQ_1, we examined the scores given to all results given by Exemplar during the user study. We also consider the Java programmers' confidence level rankings of those results. The programmers ranked results using a four-level Likert scale (see Section 5.1). We grouped Exemplar's scores for applications by the confidence level provided by the case study participants for those applications. Fig. 4 is a statistical summary of the scores for the results, grouped by the confidence level. These scores were obtained from Exemplar using all 209 queries that the users produced for 22 tasks during the case study.[12] We have made all these results available for download from the Exemplar website so that other researchers can reproduce our analysis and the results.

7.1.1 Hypotheses for RQ_1

We want to determine to what degree the mean of the scores from Exemplar increase as the user confidence level rankings increase. We introduce the following null and alternative hypotheses to evaluate the significance of any difference at a 0.05 level of confidence:

- $H_{10\text{-}null}$: The null hypothesis is that there is no difference in the values of Exemplar scores of applications among the groupings by the confidence level.
- $H_{10\text{-}alt}$: An alternative hypothesis to $H_{10\text{-}null}$ is that there is a statistically significant difference in the values of Exemplar scores of applications among the groupings by the confidence level.

7.1.2 Testing the Null Hypothesis

The results of ANOVA for $H_{10\text{-}null}$ confirm that there are statistically significant differences among the groupings by confidence level. Intuitively, these results mean that higher scores imply higher confidence levels from programmers. Higher confidence levels, in turn, point to higher relevance (see Section 5). Table 6 shows the F-value, P-value, and critical F-value for the variance among the groups. We reject the null hypothesis $H_{10\text{-}null}$ because the $F > F_{critical}$. Additionally, $P < 0.05$. Therefore, we find evidence supporting the alternative hypothesis $H_{10\text{-}alt}$.

Finding supporting evidence for $H_{10\text{-}alt}$ suggests that we can answer RQ_1. To confirm these results, however, we grouped the results in terms of relevant (e.g., confidence 3 or 4) and nonrelevant (e.g., confidence 1 or 2), and tested the difference of these groups. A randomization test of these groups showed a P-value of < 0.0001, which provides further evidence for answering RQ_1. Therefore, we find that higher Exemplar scores do in fact match to higher confidence level rankings from participants in the user study.

7.2 Principal Components of the Score

The relevance score that Exemplar computes for every retrieved application is actually a combination of the three metrics (WOS, RAS, and DCS) presented in Section 3. Technically, these three metrics were added together with equal weights, using an affine transformation during the case study. Ideally, each of these metrics should contribute orthogonal information to the final relevance score, meaning that each metric will indicate the relevance of applications when the others might not. To analyze the degree to which WOS, RAS, and DCS contribute orthogonal information to the final score and to address RQ_2, we used Principal Component Analysis (PCA)[24]. PCA locates uncorrelated dimensions in a dataset and connects input parameters to these dimensions. By looking at how the inputs connect to the principal components, we can deduce how each component relates to the others.

To apply PCA, we ran Exemplar using the queries from the case study and obtained WOS, RAS, DCS, and combined scores for the top 10 applications for each of the queries. We then used these scores as the input parameters to be analyzed. PCA identified three principal components;

12. Note that the participants only completed 22 out of 26 total tasks available.

TABLE 4
Factor Loading through Principal Component Analysis of Each of the Scores (WOS, RAS, and DCS) that Contribute to the Final Score in Exemplar (ALL)

	PC1	PC2	PC3
Proportion	43.8%	31.5%	24.8%
Cumulative	43.8%	75.3%	100%
WOS	-0.730	0.675	0.106
RAS	0.995	0.091	-0.039
DCS	-0.010	-0.303	0.953
ALL	0.477	0.839	0.263

Table 4 shows the results of this analysis. We find that the first principal component is primarily RAS (99.5 percent association), the second component is somewhat linked to WOS (67.5 percent association), and the third component is primarily DCS (95.3 percent association). The final Exemplar score (denoted ALL) is linked to each of the primary components, which we expect because the input parameters combine to form the Exemplar score. Because WOS, RAS, and DCS are all positively associated with their own principal components, we conclude that each metric provides orthogonal information to Exemplar.

We also computed the Spearman correlations [43] for each input parameter to each other. These correlations are presented in Table 5. WOS and RAS are negatively correlated with one another, a fact suggesting that the two metrics contribute differently to the final ranking score. Moreover, RAS exhibits moderate correlation to the final Exemplar score, while WOS is at least positively correlated. DCS, however, is entirely uncorrelated with either RAS or WOS. We draw two conclusions given these results. First, we answer RQ_2 by observing that RAS and WOS do capture orthogonal information (see PCA results in Table 4). Second, because DCS does not correlate with the final score and because DCS did not appear to benefit Exemplar during the case study (see Section 6.4), we removed DCS from Exemplar. We do not consider DCS in any other analysis in this section.

TABLE 5
Spearman Correlations of the Score Components to Each Other and to the Final Ranking

	WOS	RAS	DCS	ALL
WOS	1	-0.741	-0.104	0.142
RAS	-0.741	1	-0.046	0.482
DCS	-0.104	-0.046	1	-0.005
ALL	0.142	0.482	-0.005	1

7.2.1 Analysis of WOS and RAS

Given that WOS and RAS contribute orthogonally to the Exemplar score, we now examine whether combining them in Exemplar returns more relevant applications versus each metric individually. We judged the benefit of WOS and RAS by computing each metric for every application using the queries from the case study. We then grouped both sets of scores by the confidence level assigned to the application by the case study participants in a setup similar to that in Section 7.1. Figs. 5a and 5b are statistical summaries for the WOS and RAS scores, respectively. We introduce the following null and alternative hypotheses to evaluate the significance of any difference at a 0.05 level of confidence:

- $H_{11\text{-}null}$: The null hypothesis is that there is no difference in the values of WOS scores of applications among the groupings by confidence level.
- $H_{11\text{-}alt}$: An alternative hypothesis to $H_{11\text{-}null}$ is that there is a statistically significant difference in the values of WOS scores of applications among the groupings by confidence level.
- $H_{12\text{-}null}$: The null hypothesis is that there is no difference in the combined values of RAS scores of applications among the groupings by confidence level.
- $H_{12\text{-}alt}$: An alternative hypothesis to $H_{12\text{-}null}$ is that there is a statistically significant difference in the values of RAS scores of applications among the groupings by confidence level.

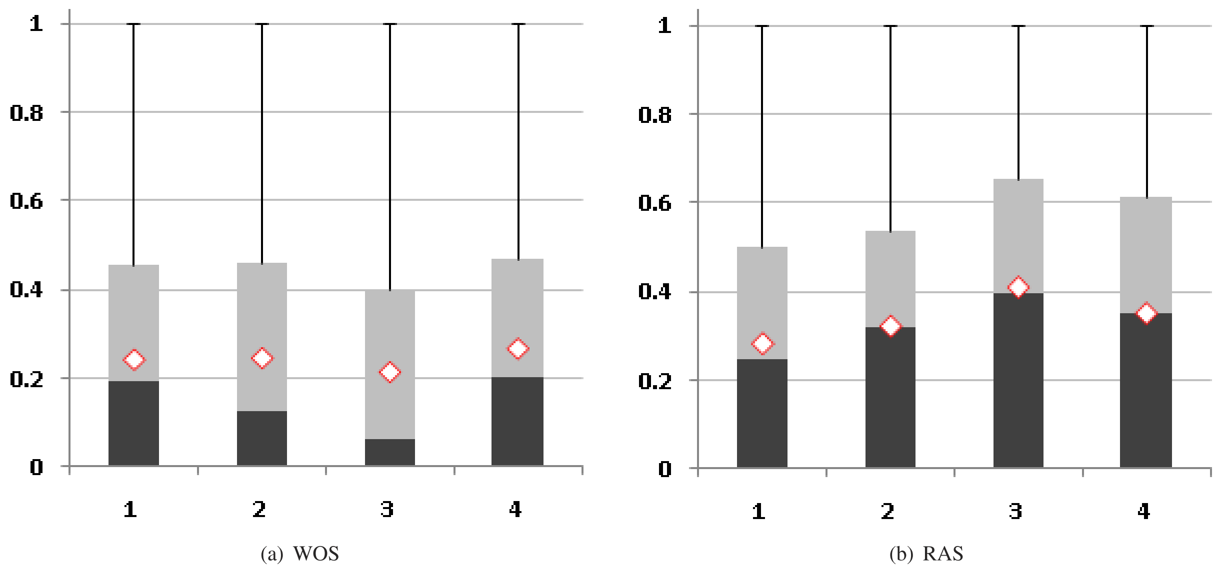


Fig. 5. Statistical summary of the WOS and RAS scores from the case study of Exemplar.

TABLE 6
Results of Testing $H_{10\text{-}null}$, $H_{11\text{-}null}$, and $H_{12\text{-}null}$

	F	P	$F_{critical}$
$H_{10-null}$	12.31	6E-08	2.61
$H_{11-null}$	1.97	0.12	2.61
$H_{12-null}$	8.18	2E-05	2.61

7.2.2 Testing the Null Hypotheses

We used one-way ANOVA to evaluate $H_{11\text{-}null}$ and $H_{12\text{-}null}$ that the variation in the experiment is no greater than that due to normal variation of the case study participants choices of confidence level as well as chance matching by WOS and RAS, respectively. The results of ANOVA confirm that there are statistically significant differences among the groupings by confidence level for RAS, but not for WOS. Table 6 shows the F-value, P-value, and critical F-value for the variance among the groups for WOS. Table 6 shows the same values for RAS. We do not reject the null hypothesis $H_{11\text{-}null}$ because $\mathrm{F} < \mathrm{F}_{critical}$. Additionally, $\mathrm{P} > 0.05$. Therefore, we cannot support the alternative hypothesis $H_{12\text{-}alt}$. On the other hand, we reject the null hypothesis $H_{12\text{-}null}$ because the $\mathrm{F} > \mathrm{F}_{critical}$. Additionally, $\mathrm{P} < 0.05$. Therefore, we find evidence supporting the alternative hypothesis $H_{12\text{-}alt}$.

We finish our study of the contributions of RAS, WOS, and DCS by concluding that RAS improves the results by a statistically significant amount. Meanwhile, we cannot infer any findings about WOS because we could not reject $H_{11\text{-}null}$. We did observe specific instances in the case study where WOS contributed to the retrieval of relevant results when RAS did not (see Section 9). Therefore, we include WOS in the final version of Exemplar, albeit with a weight reduced by 50 percent, from 0.5 to 0.25. We also increased the weight of RAS by 50 percent from 0.5 to 0.75 because we found that RAS contributes to more relevant results than WOS.

7.3 Keyword Sensitivity of Exemplar

Recent research shows that users tend to generate different kinds of queries [3]. It may be the case that different users of Exemplar create different queries which represent the same task that those users need to implement. If this occurs, some users may see relevant results, whereas others see irrelevant ones. During the case study, we provided the participants with 22 varied tasks. The participants were then free to read the tasks and generate queries on their own. Exemplar may retrieve different results for the same task given different queries, even if the participants generating those queries all interpreted the meaning of the task in the same way. This presents a threat to validity for the case study because different participants may see different results (and produce different rankings) for the same task. For example, consider Task 1 from Section 5.5. Table 7 shows two separate queries generated independently by users during the case study for this task.[13] By including more keywords, the author of the second query found three different applications than the author of the first query. In this section, we will answer RQ_3 by studying how sensitive Exemplar is to variations in the query as formulated by different users for the same task.

13. We generated the results in Table 7 using Exemplar in the same configuration as in the case study, which can be accessed here: http://www.xemplar.org/original.html (verified 03/28/2011).

TABLE 7
The Top 10 Applications Returned by Exemplar for Two Separate Queries

	"sound voice midi"	"sound voice audio midi connection gui"
1	Tritonus	Tritonus
2	Java Sound Res	RasmusDSP
3	RasmusDSP	Audio Develop
4	TuxGuitar	TuxGuitar
5	MidiQuickFix	MidiQuickFix
6	Audio Develop	Java Sound Res
7	FluidGUI	RPitch
8	DGuitar	DGuitar
9	Cesar	Music and Audio
10	Saiph	JVAPTools

Both queries were generated by users during the case study while reading the same task. Shaded cells indicate applications in both sets of results. Application names in bold were rated with confidence level 3 or 4 (relevant or highly relevant) by the author of the associated query. Note: Ties of relevance scores are broken randomly; applications with identical scores may appear in a different order.

First, we need to know how different the queries and the results are for individual tasks. We computed the *query overlap* to measure how similar queries are for each task. We defined query overlap as the pairwise comparison of the number of words which overlap for each query. The formula is $queryoverlap = \frac{|query_1 \bigcap query_2|}{|query_1 \bigcup query_2|}$ where `query1` is the set of words is the first query and `query2` is the set of words in the second query. For example, consider the queries "sound voice midi" and "sound voice audio midi connection gui." The queries share the words "sound," "voice," and "midi." The total set of words is "sound voice midi audio connection gui." Therefore, the query overlap is 0.5, or 50 percent. To obtain the query overlap for a task, we simply computed the overlap numbers for every query to every other query in the task. The queries were processed in the same way as they are in Exemplar; we did not perform stemming or removal of stop words.

Because we see different queries for each task, we expect to see different sets of results from Exemplar over a task. We surmise that if two users give two different queries for the same task, then Exemplar will return different results as well. We want to study the degree to which Exemplar is sensitive to changes in the query for a task. Therefore, we calculate the *results overlap* for each task using the formula $resultsoverlap = \frac{|unique-total|}{|expected-total|}$ where `total` is the total number of results found for a given task, `unique` is the number of those results which are unique, and `expected` is the number of results we expect if all the results overlapped (e.g., the minimum number of unique results possible). For example, consider the situation in Table 7 where, for a single task, two users created two different queries. In the case study, participants examined the top 10 results, meaning that Exemplar returned 20 total results. At least 10 of the results must be unique, which is the expected number if Exemplar returned the same set for all three

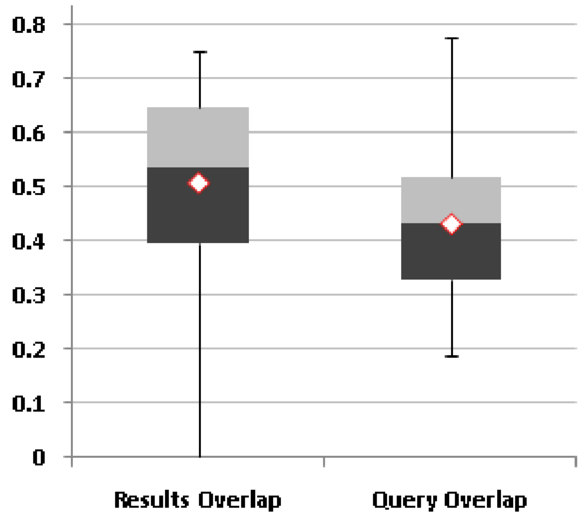


Fig. 6. Statistical summary of the overlaps for tasks. The x-axis is the type of overlap. The y-axis is the value of the overlap.

queries. In Table 7, however, 13 of the results were unique; results overlap would be 0.7, or 70 percent overlapped.

Statistical summaries of the results overlap and query overlap are in Fig. 6. The Spearman correlation for the overlaps was 0.356. We observe a weak correlation between results and query overlap, which we expect because more similar queries will most likely cause Exemplar to produce more similar results. Therefore, to answer RQ_3, we do find evidence that Exemplar is sensitive to differences in the queries, even if those queries were created to address the same task.

7.4 Sensitivity to the Number of API Calls

The RAS component of Exemplar is responsible for ranking applications based on the API calls made in those applications. This component first locates a number of descriptions of API calls which match the keywords provided in the user's query. It then matches those API calls to applications which use those calls. During the case study, we limited the number of API calls that RAS considers to 200 due to performance overhead. In this section, we analyze the effect this design decision had on the search results.

The maximum number of APIs to consider is an internal parameter to Exemplar called $maxapi$. To study its effects, we first obtained all 209 queries written by participants in the case study from Section 5. We then set $maxapi$ to infinity (so that potentially every API could be returned) and ran every query through Exemplar. From this run, we determined that the maximum number of API calls extracted for any query was 406. We also stored the list of results from this run.

We then ran Exemplar with various entries as input for $maxapi$ ranging between 1 and 406.[14] We then calculated the *results overlap* for the results of each of these runs against the results from the run in which $maxapi$ was set to infinity. In this way, we computed the percent of overlap of the various levels of $maxapi$ with case in which all API calls are considered. The results of this analysis are summarized in Fig. 7. We observe that when $maxapi$ is set to a value greater than or equal to 200, the percent overlap is always above 80 percent, meaning that 80 percent of the results are identical to those in the case when all API calls are considered. We set $maxapi$ to 200 in the remainder of this paper.

14. Note that Exemplar produces the same results when $maxapi$ is set to 406 and infinity since 406 was the maximum amount of API calls returned.

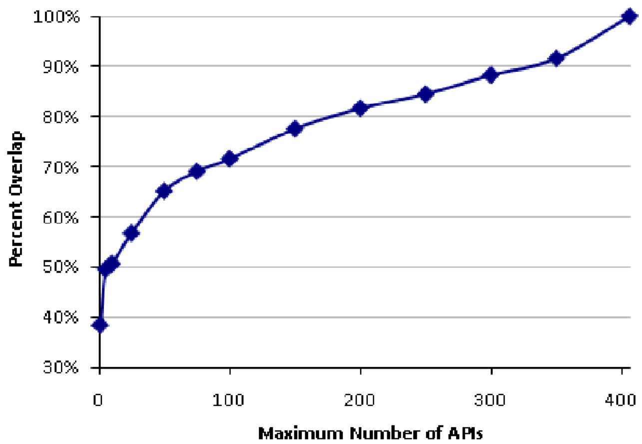


Fig. 7. A chart of the results overlap from various levels of $maxapi$. The x-axis is the value of the overlap. The y-axis is the value of $maxapi$.

7.5 Sensitivity to Frequency of API Calls

The RAS component ranking considers the frequency of each API call that occurs in each application. For example, if an application A makes an API call c twice, and an application B makes an API call c only once, and c is determined to be relevant to the user query, then application A will be ranked higher than B. In Exemplar, we use static analysis to determine the API calls used by an application. Therefore, we do not know the precise number of times an API call is actually made in each application because we do not have execution information for these applications. For example, consider the situation where application A calls c twice and B calls c once. If the call to c in B occurs inside a loop, B may call c many more times than A, but we will not capture this information.

We developed a binary version of RAS to study the effects this API frequency information may cause in our case study. The binary version of RAS does not consider the frequency of each API call in the applications. More formally, the binary RAS calculates the scores according to the formula $S^j_{ras} = \frac{\sum_{i=1}^{p} C^j_i}{|A|^j}$, where $|A|^j$ is the total number of API calls in the application j and p is the number of API calls retrieved for the query.

We then executed Exemplar using the 209 queries from the case study in Section 5 for both the binary version of RAS and the RAS that considers frequencies of API calls as described in Section 3.3. We computed the *results overlap* between the results for both. The mean overlap for the results of every query was 93.2 percent. The standard deviation was 13.4 percent. Therefore, we conclude that the results from Exemplar with the binary version of RAS are not dramatically different from the frequency-based version of RAS. We use the frequency-based version of RAS in the remainder of this paper.

TABLE 8
Plan for the Case Study of Exemplar_{NEW} and Exemplar_{OLD}

Experiment	Group	Search Engine	Task Set
1	G1	NEW	T1
	G2	OLD	T2
2	G1	OLD	T2
	G2	NEW	T1

8 Evaluation of Changes to Exemplar

We made several alterations to Exemplar based on our analysis in Section 7. Specifically, we removed DCS, rebalanced the weights of WOS and RAS (to 0.25 and 0.75), and updated the interface so that project source code is visible without downloading whole projects. We compare the quality of the results from the updated version of Exemplar against the previous version. In this study, we refer to the previous Exemplar as Exemplar_{OLD} and the new Exemplar as Exemplar_{NEW}.

8.1 Methodology

We performed a case study identical in design to that presented in Section 5, except that we evaluate two engines (Exemplar_{NEW}, Exemplar_{OLD}) instead of three (EWN, END, SF). Table 8 outlines the study. We chose END to represent the old Exemplar because END was the best-performing configuration. In this case, we randomly divided 26 case study participants[15] into two groups. There were two experiments, and both groups participated in each. In each experiment, each group was given a different search engine (e.g., Exemplar_{NEW} or Exemplar_{OLD}) and a set of tasks. The participants then generated queries for each task and entered those queries into the specified search engine. The participants rated each result on a four-point Likert scale as in Section 5. From these ratings, we computed the three measures confidence (C), precision (P), and normalized discounted cumulative gain (NG).

8.2 Hypotheses

We introduce the following null and alternative hypotheses to evaluate the differences in the metrics at a 0.05 confidence level:

- H_{13}: The null hypothesis is that there is no difference in the values of C for Exemplar_{NEW} versus Exemplar_{OLD}. Conversely, the alternative is that there is a statistically significant difference in the values of C for Exemplar_{NEW} versus Exemplar_{OLD}.
- H_{14}: The null hypothesis is that there is no difference in the values of P for Exemplar_{NEW} versus Exemplar_{OLD}. Conversely, the alternative is that there is a statistically significant difference in the values of P for Exemplar_{NEW} versus Exemplar_{OLD}.
- H_{15}: The null hypothesis is that there is no difference in the values of NG for Exemplar_{NEW} versus Exemplar_{OLD}. Conversely, the alternative is that there is a statistically significant difference in the values of NG for Exemplar_{NEW} versus Exemplar_{OLD}.

15. Nine of the participants in this study were graduate students from the University of Illinois at Chicago. Five were graduate students at the College of William & Mary. Ten were undergraduate students at William & Mary. We reimbursed the participants $35 after the case study.

8.3 Results

We applied randomization tests to evaluate the hypotheses H_{13}, H_{14}, and H_{15}. The results of this test are in Table 9. We do not reject the null hypothesis H_{14} because the P-value is greater than 0.05. Therefore, participants do not report a statistically significant difference in terms of precision of the results. On the other hand, we reject the null hypotheses H_{13} and H_{15}, meaning that participants report higher confidence level in the results. Also, the participants report higher normalized discounted cumulative gain when using Exemplar_{NEW} versus Exemplar_{OLD}.

The difference in average confidence level between the updated and original versions of Exemplar is statistically significant, as seen in Fig. 8a, though the difference is very small. The difference in precision is not statistically significant (see Fig. 8b). One explanation for the small size of this difference is that both versions of Exemplar return the same sets of applications to the user. Returning the same set of applications is expected because both Exemplar_{NEW} and Exemplar_{OLD} use the same underlying information to locate these applications (e.g., API calls and project descriptions). The order of the results is also important, and the new version of Exemplar does return the more relevant results in higher positions, as reported by the normalized discounted cumulative gain (NG, see Fig. 8c).

Table 10 illustrates an example of the improvement made by Exemplar_{NEW}. This table includes the results for the same query on both engines as well as the confidence level for the applications as reported by a participant in the case study. The normalized discounted cumulative gain is higher in this example for Exemplar_{NEW} than Exemplar_{OLD}. Even though a majority of the applications are shared by both sets of results, Exemplar_{NEW} organizes the results such that the most-relevant applications appear sooner.

8.4 Participant Comments on Exemplar_{NEW}

Seventeen of the case study participants answered the same exit survey from Table 3. The responses generally support those which we discuss in Section 6.5: Roughly half of the participants reported rarely or never using source code search engines, and of those a majority prefer to use Google. The top reason cited for not using source code search engines was the perceived poor quality results given by those engines. These results, along with those in Section 6.5, are a strong motivation for improvements in source code search engines.

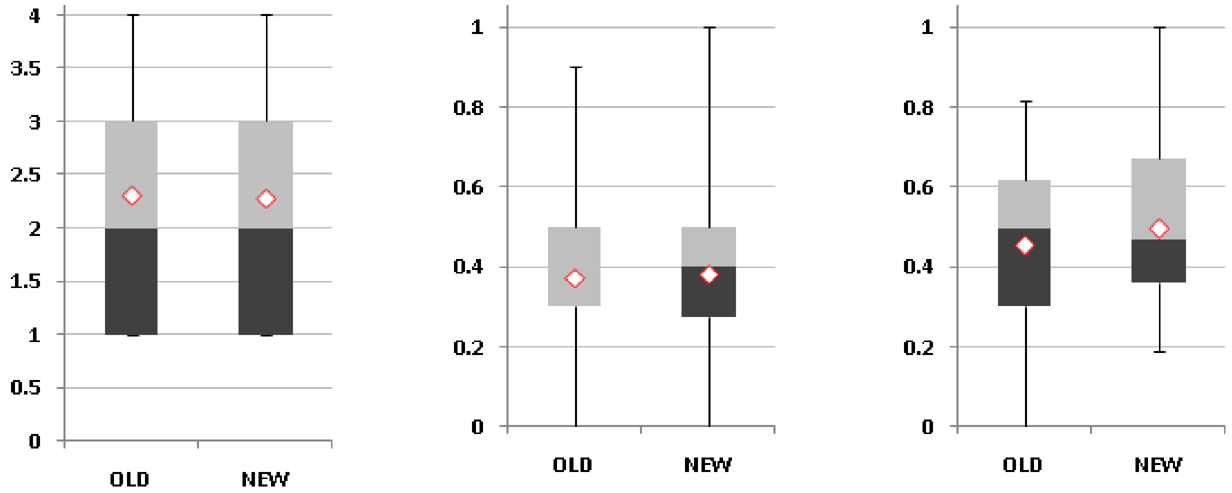
In addition to rebalancing the weights of the ranking components in Exemplar_{NEW}, we made the source code of the applications immediately available through the engine. The following are comments provided by participants regarding these changes. We conclude from these comments that 1) users prefer to see source code along with relevant applications, and 2) API calls helped participants determine the relevance of results.

- "Very convenient to be able to open to view source files immediately. Much more convenient to user."
- "[WOS in Exemplar_{OLD}] got in the way quite a bit"
- "I definitely like viewing code in the browser better"
- "[Exemplar_{NEW}] is really useful since we can know which API we should choose."

TABLE 9
Results of Randomization Tests of Hypotheses, H, for Dependent Variable Specified in the Column Var (C, P, or NG) Whose Measurements Are Reported in the Following Columns

H	Var	Approach	Samples	Min	Max	Median	μ	C	p
H_{13}	C	Exemplar$_{NEW}$	556	1	4	2	2.27	0.05	0.00156
		Exemplar$_{OLD}$	556	1	4	2	2.30		
H_{14}	P	Exemplar$_{NEW}$	40	0	1.00	0.40	0.38	-0.15	0.23738
		Exemplar$_{OLD}$	40	0	0.90	0.30	0.37		
H_{15}	NG	Exemplar$_{NEW}$	40	0.19	1.00	0.47	0.50	-0.15	0.04507
		Exemplar$_{OLD}$	40	0	0.82	0.49	0.46		

Extremal values, median, means, μ, and the Pearson correlation coefficient, C, are reported along with the results of the evaluation of the hypotheses, i.e., statistical significance, p.



(a) Confidence level, C. (b) Precision, P. (c) Normalized Discounted Gain, NG.

Fig. 8. Statistical summary of C, P, and NG from the case study evaluating the new version of Exemplar. The y-axis is the value for C, P, or NG from the case study. The x-axis is the version of Exemplar.

- "[API calls] are very useful if the call is relevant, a lot of API calls had nothing to do with the task."
- "[API calls] are very useful for determining initial area of source code which should be examined."

TABLE 10
The Search Results from a Single Query from the Second Case Study; Applications Are Listed with the Assigned Confidence Levels

"glyph painting"			
Exemplar$_{OLD}$		Exemplar$_{NEW}$	
Jazilla	1	Jazilla	1
DrawSWF	4	DrawSWF	4
Image inpainting	1	McBilliards	3
SandboxPix	1	Waba for Dos	3
McBilliards	3	BioGeoTools	1
Waba for Dos	3	TekMath	2
BioGeoTools	1	SWTSwing	0
TekMath	2	Java2C	0
SWTSwing	0	JSpamAssassin	0
DESMO-J	0	netx	0
NG Top 6	0.5143		0.5826
NG Top 10	0.4247		0.4609

A case study participant generated the query and provided the relevancy rankings when evaluating Exemplar$_{OLD}$. Applications with a confidence level zero were not able to be accessed by the participant, and are discarded during our analysis. We ran the same query on Exemplar$_{NEW}$. The confidence levels for the results of Exemplar$_{NEW}$ are copied from the confidence levels given by the participant who ran Exemplar$_{OLD}$. NG represents the normalized discounted cumulative gain for the top 6 (all evaluated, zeros discarded) and top 10 (all retrieved, zeros included).

8.5 Suggestions for Future Work

The participants in the case study had several suggestions for Exemplar, and we have incorporated these into our future work. One participant asked that we filter "trivial" results such as API calls named `equal()` or `toString()`. Another suggested that we provide descriptions of API calls directly on the results page. A participant also requested a way to sort and filter the API calls; he was frustrated that some source code files contain "the same type-check method many times."

9 Supporting Examples

Table 11 shows the results from Exemplar for three separate queries, including the top 10 applications and the WOS and RAS scores for each.[16] For instance, consider the query *connect to an http server*. Only one of the top 10 results from Exemplar is returned (see Table 11) due to a high WOS score (e.g., because the query matches the high-level description of the project). The remaining nine projects pertain to different problem domains, including Internet security testing, programming utilities, and bioinformatics. These nine applications, however, all use API calls from the Java class java.net.HttpURLConnection.[17] Exemplar was

16. We generated the results in Table 11 using Exemplar in the same configuration as in the case study, which can be accessed here: http://www.xemplar.org/original.html.

17. The documentation for this API class can be found at: http://download.oracle.com/javase/6/docs/api/java/net/HttpURLConnection.html (verified 03/28/2011).

TABLE 11
The Top 10 Applications Returned by Exemplar for Three Separate Queries, along with the WOS and RAS Scores for Each

	"connect to http server"			"text editor"			"find replace string text files"		
	Application	WOS	RAS	Application	WOS	RAS	Application	WOS	RAS
1	DataShare	100%	0%	jeHep	52%	89%	RText	91%	0%
2	X4technology	0%	100%	XNap Commons	0%	100%	Nodepublisher	0%	66%
3	jpTools	0%	96%	SWediT	92%	0%	XERP	44%	18%
4	JMS for j2ms	0%	96%	Plugins jext	87%	0%	J	54%	0%
5	MicroEmulator	0%	96%	PalmEd	87%	0%	j-sand	53%	0%
6	ReadSeq bioinfo	0%	95%	PowerSwing	0%	85%	DocSearch	48%	0%
7	httpunit	0%	95%	Graveyard	83%	0%	MMOpenGraph	43%	0%
8	WebCQ	0%	95%	JavaTextEditor	82%	0%	AppletServer	0%	41%
9	WebXSSDetector	0%	95%	Eclipse Edit	81%	0%	MultiJADS	0%	39%
10	Organism System	0%	90%	Comic book edit	65%	15%	GalleryGrabber	0%	39%

The DCS score was zero in every case. Note: Ties of relevance scores are broken randomly; applications with identical scores may appear in a different order.

able to retrieve these applications only because of the contribution from the RAS score.

Other queries may reflect the high-level concepts in a software application, rather than low-level details. For example, for the query *text editor*, Exemplar returns six of 10 top results without any matching from RAS (see Table 11). While the query does match certain API calls, such as those in the class javax.swing.text.JTextComponent,[18] Exemplar finds several text editing programs which do not use API calls from matching documentation. Locating these applications was possible because of relatively high WOS scores.

We observed instances during the case study where the negative correlation between WOS and RAS improved the final search results. Consider Task 2 from Section 5.5. For this task, one programmer entered the query *find replace string text files* into Exemplar (see Table 11). The first result was a program called RText, which is a programmer's text editor with find/replace functionality. The second result was Nodepublisher, a content management system for websites. Nodepublisher's high-level description did not match the query and has a WOS score of 0 percent. The query did match several API call descriptions, including calls inside the class java.text.DictionaryBasedBreakIterator,[19] which Nodepublisher uses. Conversely, RText contained no API calls with documentation matching the query, but had a relevant high-level description. Since both applications were rated as highly relevant by the programmer in the case study, both WOS and RAS aided in finding a relevant result for this query. Specific situations such as this one support our decision to keep WOS in the final version of Exemplar, even with a reduced weight (see Section 7.2.2). Not all applications with high WOS or RAS scores were relevant, however. Despite occurring in the top 10 list of applications, both MMOpenGraph and AppletServer were rated with a confidence level of 2 ("mostly irrelevant") by the author of the query.

10 Related Work

Different code mining techniques and tools have been proposed to retrieve relevant software components from different repositories as is shown in Table 12. CodeFinder iteratively refines code repositories in order to improve the precision of returned software components [16]. Codefinder finds similar code using spreading activation based on the terms that appear in that code. Exemplar is different in that we locate source code based on keywords from API documentation. It is not necessary for Exemplar to find any matching keywords in the source code itself.

The Codebroker system uses source code and comments written by programmers to query code repositories to find relevant artifacts [50]. Unlike Exemplar, Codebroker is dependent upon the descriptions of documents and meaningful names of program variables and types, and this dependency often leads to lower precision of returned projects.

Even though it returns code snippets rather than applications, Mica is similar to Exemplar since it uses help pages to

TABLE 12
Comparison of Exemplar with Other Related Approaches

Approach	Granularity		Corpora	Query Expansion
	Search	Input		
CodeFinder [16]	M	C	D	Yes
CodeBroker [51]	M	C	D	Yes
Mica [45]	F	C	C	Yes
Prospector [29]	F	A	C	Yes
Hipikat [9]	A	C	D,C	Yes
xSnippet [39]	F	A	D	Yes
Strathcona [19][20]	F	C	C	Yes
AMC [17]	F	C	C	No
Google Code	F,M,A	C,A	D,C	No
Sourceforge	A	C	D	No
SPARS-J [22][23]	M	C	C	No
Sourcerer [27]	F,M,A	C	C	No
Sourcerer API Search [4]	F	C,A	C	No
CodeGenie [26]	F,M	T	C	No
SpotWeb [47]	M	C	C	Yes
ParseWeb [48]	F	A	C	Yes
S^6 [36]	F	C,A,T	C	Manual
Krugle	F,M,A	C,A	D,C	No
Koders	F,M,A	C,A	D,C	No
SNIFF [8]	F,M	C,A	D,C	Yes
Blueprint [7]	F	C,A	C	No
Exemplar [15]	F,M,A	C,A	D,C	No

*Column `Granularity` specifies how search results are returned by each approach (**F**ragment of code, **M**odule, or **A**pplication), and how users specify queries (**C**oncept, **A**PI call, or **T**est case). The column `Corpora` specifies the scope of search, i.e., **C**ode or **D**ocuments, followed by the column `Query Expansion` that specifies if an approach uses this technique to improve the precision of search queries.*

18. The documentation for this API class can be found at: http://cupi2.uniandes.edu.co/site/images/recursos/javadoc/j2se/1.5.0/docs/api/javax/swing/text/JTextComponent.html (verified 03/28/2011).

19. The documentation for this API class can be found at: http://www.docjar.com/docs/api/java/text/DictionaryBasedBreakIterator.html (verified 03/28/2011).

find relevant API calls to guide code search [45]. However, Mica uses help documentation to refine the results of the search, while Exemplar uses help pages as an integral instrument in order to expand the range of the query.

SSI examines the API calls made in source code in order to determine the similarity of that code [5]. SSI indexes each source code element based on the identifier names and comments in that code. Then, SSI adds terms to the index of a source element. The new terms come from other source code elements which use the same set of API calls. Additionally, SSI seeds the index with keywords from API call documentation. On the other hand, Exemplar matches query keywords directly to API documentation, and then calculates RAS, which is a ranking based on which projects use the API calls that the matching documentation describes. The fundamental difference between Exemplar and SSI is that Exemplar bases its ranking on how many relevant API calls appear in the source code (RAS, Section 3.3), unlike SSI, which ranks source code based on the keyword occurrences in the source code. Also, Exemplar has been evaluated with a user study of professional programmers.

SNIFF extends the idea of using documentation for API calls for source code search [14], [45] in several ways [8]. After retrieving code fragments, SNIFF then performs intersection of types in these code chunks to retain the most relevant and common part of the code chunks. SNIFF also ranks these pruned chunks using the frequency of their occurrence in the indexed code base. In contrast to SNIFF [8], MICA [45], and our original MSR idea [14], we evaluated Exemplar using a large-scale case study with 39 programmers to obtain statistically significant results, we followed a standard IR methodology for comparing search engines, and we return fully executable applications. Exemplar's internals differ substantially from previous attempts to use API calls for searching, including SNIFF: Our search results contain multiple levels of granularity, we conduct a thorough comparison with the state-of-the-art search engine using a large body of Java application code, and we are not tied to a specific IDE.

Prospector is a tool that synthesizes fragments of code in response to user queries that contain input types and desired output types [29]. Prospector is an effective tool to assist programmers in writing complicated code; however, it does not provide support for a full-fledged code search engine.

Keyword programming is a technique which translates a few user-provided keywords into a valid source code statement [28]. Keyword programming matches the keywords to API calls and the parameters of those calls. Then, it links those parameters to variables or other functions also mentioned in the keywords. Exemplar is similar to keyword programming in that Exemplar matches user queries to API calls and can recommend usage of those calls. Unlike keyword programming, Exemplar show examples of previous usage of those APIs and does not attempt to integrate those calls into the user's own source code.

The Hipikat tool recommends relevant development artifacts (i.e., source revisions associated with a past change task) from a project's history to a developer [9]. Unlike Exemplar, Hipikat is a programming task-oriented tool that does not recommend applications whose functionalities match high-level requirements.

Strathcona is a tool that heuristically matches the structure of the code under development to the example code [19], [18]. Strathcona is beneficial when assisting programmers while working with existing code; however, its utility is not applicable when searching for relevant projects given a query containing high-level concepts with no source code.

There are techniques that navigate the dependency structure of software. Robillard proposed an algorithm for calculating program elements of likely interest to a developer [37], [38]. FRAN is a technique which helps programmers to locate functions similar to given functions [41]. Finally, XSnippet is a context-sensitive tool that allows developers to query a sample repository for code snippets that are relevant to the programming task at hand [39]. Exemplar is similar to these algorithms in that it uses relations between API calls in the retrieved projects to compute the level of interest (ranking) of the project. Unlike these approaches, Exemplar requires only a natural language query describing a programming task. We found in this paper that considering the dataflow among API calls does not improve the relevancy of results in our case.

Existing work on ranking mechanisms for retrieving source code are centered on locating components of source code that match other components. Quality of match (QOM) ranking measures the overall goodness of match between two given components [46], which is different from Exemplar which retrieves applications based on high-level concepts that users specify in queries. *Component rank model (CRM)* is based on analyzing actual usage relations of the components and propagating the significance through the usage relations [22], [23]. Yokomori et al. used CRM to measure the impact of changes to frameworks and APIs [52]. Unlike CRM, Exemplar's ranking mechanism is based on a combination of the usage of API calls and relations between those API calls that implement high-level concepts in queries.

S^6 is a code search engine that uses a set of user-guided program transformations to map high-level queries into a subset of relevant code fragments [36], not complete applications. Like Exemplar, S^6 returns source code; however, it requires additional low-level details from the user, such as data types of test cases.

11 Conclusions

We created Exemplar, a search engine for highly relevant software projects. Exemplar searches among over 8,000 Java applications by looking at the API calls used in those applications. In evaluating our work, we showed that Exemplar outperformed SourceForge in a case study with 39 professional programmers. These results suggest that the performance of software search engines can be improved if those engines consider the API calls that the software uses. Also, we modified Exemplar to increase the weight of RAS, and performed a second case study evaluating the effects of this increase. We found that not only does including API call information increase the relevance of the results, but it also improves the ordering of the results. In other words, Exemplar places the relevant applications at the top of list of results.

ACKNOWLEDGMENTS

The authors thank the anonymous *TSE* and ICSE 2010 reviewers for their comments and suggestions that helped them to greatly improve the quality of this submission. They are grateful to Dr. Kishore Swaminathan, the Chief Scientist and Director of Research for his invaluable support. They also thank Malcom Gethers from William & Mary (W&M) for assisting in computation of the statistical tests, Bogdan Dit from W&M for helpful suggestions in editing this paper, and Himanshu Sharma from the University of Illinois Chicago for his work on the updated interface for Exemplar. This work is supported by US National Science Foundation (NSF) CCF-0916139, CCF-0916260, and Accenture Technology Labs. Any opinions, findings and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

REFERENCES

[1] A. Al-Maskari, M. Sanderson, and P. Clough, "The Relationship between IR Effectiveness Measures and User Satisfaction," *Proc. 30th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pp. 773-774, 2007.

[2] N. Anquetil and T.C. Lethbridge, "Assessing the Relevance of Identifier Names in a Legacy Software System," *Proc. Conf. Centre for Advanced Studies on Collaborative Research*, p. 4, 1998.

[3] S. Bajracharya and C. Lopes, "Analyzing and Mining an Internet-Scale Code Search Engine Usage Log," *J. Empirical Software Eng.*, 2009.

[4] S. Bajracharya, J. Ossher, and C. Lopes, "Searching API Usage Examples in Code Repositories with Sourcerer API Search," *Proc. ICSE Workshop Search-Driven Development: Users, Infrastructure, Tools and Evaluation*, pp. 5-8, 2010.

[5] S.K. Bajracharya, J. Ossher, and C.V. Lopes, "Leveraging Usage Similarity for Effective Retrieval of Examples in Code Repositories," *Proc. 18th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.*, pp. 157-166, 2010.

[6] T.J. Biggerstaff, B.G. Mitbander, and D.E. Webster, "Program Understanding and the Concept Assigment Problem," *Comm. ACM*, vol. 37, no. 5 pp. 72-82, 1994.

[7] J. Brandt, M. Dontcheva, M. Weskamp, and S.R. Klemmer, "Example-Centric Programming: Integrating Web Search into the Development Environment," *Proc. 28th Int'l Conf. Human Factors in Computing Systems*, pp. 513-522, 2010.

[8] S. Chatterjee, S. Juvekar, and K. Sen, "SNIFF: A Search Engine for Java Using Free-Form Queries," *Proc. Int'l Conf. Fundamental Approaches to Software Eng.*, pp. 385-400, 2009.

[9] D. Cubranic, G.C. Murphy, J. Singer, and K.S. Booth, "Hipikat: A Project Memory for Software Development," *IEEE Trans. Software Eng.* vol. 31, no. 6, pp. 446-465, June 2005.

[10] U. Dekel and J.D. Herbsleb, "Improving API Documentation Usability with Knowledge Pushing," *Proc. 31st IEEE Int'l Conf. Software Eng.*, pp. 320-330, 2009.

[11] G.W. Furnas, T.K. Landauer, L.M. Gomez, and S.T. Dumais, "The Vocabulary Problem in Human-System Communication," *Comm. ACM*, vol. 30, no. 11 pp. 964-971, 1987.

[12] M. Gabel and Z. Su, "A Study of the Uniqueness of Source Code," *Proc. 18th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.*, pp. 147-156, 2010.

[13] L.A. Granka, T. Joachims, and G. Gay, "Eye-Tracking Analysis of User Behavior in WWW Search," *Proc. 27th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pp. 478-479, 2004.

[14] M. Grechanik, K.M. Conroy, and K. Probst, "Finding Relevant Applications for Prototyping," *Proc. Fourth Int'l Symp. Mining Software Repositories*, p. 12, 2007.

[15] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C.M. Cumby, "A Search Engine for Finding Highly Relevant Applications," *Proc. 32nd ACM/IEEE Int'l Conf. Software Eng.*, pp. 475-484, 2010.

[16] S. Henninger, "Supporting the Construction and Evolution of Component Repositories," *Proc. 18th Int'l Conf. Software Eng.*, pp. 279-288, 1996.

[17] R. Hill and J. Rideout, "Automatic Method Completion," *Proc. IEEE 19th Int'l Conf. Automated Software Eng.*, pp. 228-235, 2004.

[18] R. Holmes and G.C. Murphy, "Using Structural Context to Recommend Source Code Examples," *Proc. 27th Int'l Conf. Software Eng.*, pp. 117-125, 2005.

[19] R. Holmes, R.J. Walker, and G.C. Murphy, "Strathcona Example Recommendation Tool," *Proc. 10th European Software Eng. Conf. Held Jointly with 13th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.*, pp. 237-240, 2005.

[20] R. Holmes, R.J. Walker, and G.C. Murphy, "Approximate Structural Context Matching: An Approach to Recommend Relevant Examples," *IEEE Trans. Software Eng.* vol. 32, no. 12, pp. 952-970, Dec. 2006.

[21] J. Howison and K. Crowston, "The Perils and Pitfalls of Mining Sourceforge," *Proc. Workshop Mining Software Repositories at the Int'l Conf. Software Eng.*, 2004.

[22] K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, and S. Kusumoto, "Component Rank: Relative Significance Rank for Software Component Search," *Proc. 25th Int'l Conf. Software Eng.*, pp. 14-24, 2003.

[23] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto, "Ranking Significance of Software Components Based on Use Relations," *IEEE Trans. Software Eng.* vol. 31, no. 3 pp. 213-225, Mar. 2005.

[24] I.T. Jolliffe, *Principal Component Analysis.* Springer Verlag, 1986.

[25] C.W. Krueger, "Software Reuse," *ACM Computing Surveys*, vol. 24, no. 2, pp. 131-183, 1992.

[26] O.A.L. Lemos, S. Bajracharya, J. Ossher, P.C. Masiero, and C. Lopes, "Applying Test-Driven Code Search to the Reuse of Auxiliary Functionality," *Proc. Symp. Applied Computing*, pp. 476-482, 2009.

[27] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, "Sourcerer: Mining and Searching Internet-Scale Software Repositories," *Data Mining and Knowledge Discovery*, vol. 18, pp. 300-336, 2009.

[28] G. Little and R.C. Miller, "Keyword Programming in Java," *Proc. 22nd IEEE/ACM Int'l Conf. Automated Software Eng.*, pp. 84-93, 2007.

[29] D. Mandelin, L. Xu, R. Bodik, and D. Kimelman, "Jungloid Mining: Helping to Navigate the API Jungle," *Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation*, pp. 48-61, 2005.

[30] C.D. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval.* Cambridge Univ. Press, 2008.

[31] S.S. Muchnick, *Advanced Compiler Design and Implementation.* Morgan Kaufmann Publishers Inc., 1997.

[32] G.C. Murphy, D. Notkin, and K.J. Sullivan, "Software Reflexion Models: Bridging the Gap between Source and High-Level Models," *Proc. ACM SIGSOFT Symp. Foundations of Software Eng.*, pp. 18-28, 1995.

[33] G.C. Murphy, D. Notkin, and K.J. Sullivan, "Software Reflexion Models: Bridging the Gap between Design and Implementation," *IEEE Trans. Software Eng.* vol. 27, no. 4, pp. 364-380, Apr. 2001.

[34] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval," *IEEE Trans. Software Eng.*, vol. 33, no. 6 pp. 420-432, June 2007.

[35] D. Poshyvanyk and M. Grechanik, "Creating and Evolving Software by Searching, Selecting and Synthesizing Relevant Source Code," *Proc. 31st Int'l Conf. Software Eng.—Companion Vol.*, pp. 283-286, 2009.

[36] S.P. Reiss, "Semantics-Based Code Search," *Proc. IEEE 31st Int'l Conf. Software Eng.*, pp. 243-253, 2009.

[37] M.P. Robillard, "Automatic Generation of Suggestions for Program Investigation," *Proc. 10th European Software Eng. Conf. Held Jointly with 13th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.*, pp. 11-20, 2005.

[38] M.P. Robillard, "Topology Analysis of Software Dependencies," *ACM Trans. Software Eng. Methodology*, vol. 17, no. 4, pp. 1-36, 2008.

[39] N. Sahavechaphan and K.T. Claypool, "XSnippet: Mining for Sample Code," *Proc. ACM SIGPLAN Int'l Conf. Object-Oriented Programming Systems, Languages, and Applications*, pp. 413-430, 2006.

[40] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, 1989.

[41] Z.M. Saul, V. Filkov, P. Devanbu, and C. Bird, "Recommending Random Walks," *Proc. Sixth Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. Foundations of Software Eng.,* pp. 15-24, 2007.

[42] S.E. Sim, M. Umarji, S. Ratanotayanon, and C. Lopes, "How Well Do Internet Code Search Engines Support Open Source Reuse Strategies?" *ACM Trans. Software Eng. and Methodologies,* 2009.

[43] R. Mark Sirkin, *Statistics for the Social Sciences,* third ed. Sage Publications, Aug. 2005.

[44] M.D. Smucker, J. Allan, and B. Carterette, "A Comparison of Statistical Significance Tests for Information Retrieval Evaluation," *Proc. 16 ACM Conf. Information and Knowledge Management,* pp. 623-632, 2007.

[45] J. Stylos and B.A. Myers, "A Web-Search Tool for Finding API Components and Examples," *Proc. IEEE Symp. Visual Languages and Human Centric Computing,* pp. 195-202, 2006.

[46] N. Tansalarak and K.T. Claypool, "Finding a Needle in the Haystack: A Technique for Ranking Matches Between Components," *Proc. Int'l Symp. Component-Based Software Eng.,* pp. 171-186, 2005.

[47] S. Thummalapenta and T. Xie, "Spotweb: Detecting Framework Hotspots and Coldspots via Mining Open Source Code on the Web," *Proc. 23rd IEEE/ACM Int'l Conf. Automated Software Eng.,* pp. 327-336, 2008.

[48] S. Thummalapenta and T. Xie, "Parseweb: A Programmer Assistant for Reusing Open Source Code on the Web," *Proc. IEEE/ACM 22nd Int'l Conf. Automated Software Eng.,* pp. 204-213, 2007.

[49] I.H. Witten, A. Moffat, and T.C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images,* second ed. Morgan Kaufmann, 1999.

[50] Y. Ye and G. Fischer, "Supporting Reuse by Delivering Task-Relevant and Personalized Information," *Proc. 24th Int'l Conf. Software Eng.,* pp. 513-523, 2002.

[51] Y. Ye and G. Fischer, "Reuse-Conducive Development Environments," *Automated Software Eng.* vol. 12, pp. 199-235, Apr. 2005.

[52] R. Yokomori, H. Siy, M. Noro, and K. Inoue, "Assessing the Impact of Framework Changes Using Component Ranking," *Proc. IEEE Int'l Conf. Software Maintenance,* pp. 189-198, 2009.

Collin McMillan received the MS degree in computer science from the College of William & Mary in 2009 and is working toward the PhD degree in computer science at the College of William & Mary, advised by Denys Poshyvanyk. He is a recipient of the NASA Virginia Space Grant Consortium Graduate Research Fellowship. His research interests include software engineering, software maintenance and evolution, software repository mining, and source code analysis and metrics. He is a member of the ACM and the IEEE.

Mark Grechanik received the PhD degree in computer science from the Department of Computer Sciences of the University of Texas at Austin. He is a researcher with the Accenture Technology Labs and an adjunct professor in the Departments of Computer Science at several universities, including the University of Illinois at Chicago and Northwestern University. In parallel with his academic activities, he worked for more than 20 years as a software consultant for startups and Fortune 500 companies. He is a recipient of best paper awards from competitive conferences, US National Science Foundation (NSF) grants, and patents. His research focuses on increasing programmers' productivity by automating various activities at different stages of the development life cycle. In his research, he utilizes various techniques from software engineering, language design, program analysis, and machine learning to address specific issues that affect programmers when they design, debug, and test software. His research is funded by NSF grants and industry partners who sponsor his research by investing in his ideas and providing platforms and applications to empirically validate his research prototypes. He is a member of the IEEE.

Denys Poshyvanyk received the MS and MA degrees in computer science from the National University of Kyiv-Mohyla Academy, Ukraine, and Wayne State University in 2003 and 2006, respectively, and the PhD degree in computer science from Wayne State University in 2008. He is an assistant professor at the College of William & Mary in Virginia. Since 2010, he has been serving on the steering committee of the International Conference on Program Comprehension (ICPC). He served as a program cochair for the 18th and 19th International Working Conference on Reverse Engineering (WCRE 2011 and WCRE 2012). He will also serve as a program cochair for the 21st International Conference on Program Comprehension (ICPC 2013). His research interests include software engineering, software maintenance and evolution, program comprehension, reverse engineering, software repository mining, source code analysis, and metrics. He is a member of the IEEE and the ACM.

Chen Fu received the PhD degree in computer science in 2008 from Rutgers University, under the guidance of Professor Barbara G. Ryder. His dissertation focused on cxception analysis and robustness testing of OO programs. He is a researcher at Accenture Technology Labs. His research interests include program analysis and software engineering. His recent research focuses on using program analysis techniques to improve software development and testing. The goal is to reduce manual efforts and also human error by increasing automation in these activates. He is a member of the IEEE.

Qing Xie received the BS degree in computer science in 1996 from the South China University of Technology, the MS and PhD degrees in computer science in 2002 and 2006, respectively, from the University of Maryland, College Park. She is a researcher at Accenture Technology Labs. She is a recipient of best paper awards from the International Conference of Software Testing, Verification and Validation (ICST '09) and the International Symposium on Software Reliability and Engineering (ISSRE '10). Her research interests include program testing, software engineering, software maintenance, and empirical studies. She is a member of ACM SIGSOFT, the IEEE Computer Society, and the IEEE, and has served on program committees of several international conferences and as the reviewers of reputable journals.