EXAF: A search engine for sample applications of object-oriented framework-provided concepts

Ehsan Noei, Abbas Heydarnoori*

Department of Computer Engineering, Sharif University of Technology.

ARTICLE INFO

Article history:
Received 14 November 2015
Revised 26 March 2016
Accepted 28 March 2016
Available online 14 April 2016

Keywords:
Object-oriented application frameworks
Framework-provided concepts
Sample applications
Frameworks comprehension
Code search engines

ABSTRACT

Context: Object-oriented application frameworks, such as *Java Swing*, provide reusable code and design for implementing domain-specific concepts, such as *Context Menu*, in software applications. Hence, use of such frameworks not only can decrease the time and the cost of developing new software applications, but also can increase their maintainability. However, the main problems of using object-oriented application frameworks are their large and complex APIs, and often incomplete user manuals. To mitigate these problems, developers often try to learn how to implement their desired concepts from available sample applications. Nonetheless, this introduces another hard and time-consuming challenge which is finding proper sample applications.

Objective: To address this difficulty, we introduce *EXAF* (*EXample Applications Finder*) that helps developers find sample applications which implement their desired framework-provided concepts.

Method: The majority of existing framework comprehension approaches can only help programmers to get familiar with the usage of particular fine-grained API elements of the desired framework such as its classes and methods. Nevertheless, our approach is able to find sample applications that implement a particular framework-provided concept. To this end, EXAF benefits from the *Latent Semantic Indexing* (*LSI*) information retrieval technique. We evaluated the approach using 24 concepts on top of the Microsoft .Net, Qt, and Java Swing frameworks.

Results: Based on our evaluations, the precision of EXAF is more than 79%. Besides, it can find some sample applications that could not be found by common code search engines such as the ones which are used in *SourceForge* and *Google Code*.

Conclusions: The results of our evaluations indicate that EXAF is effective in practice, and yields better search results because it considers various artifacts of a project like user reviews and bug reports.

1. Introduction

Software reuse is the use of existing software or knowledge to build new applications [1]. This can help developers to increase the quality of their software systems and to reduce the costs of software development [2]. Object-oriented application frameworks, such as *Eclipse* and *.Net*, can enable the reuse of both code and design [3]. Frameworks provide *domain-specific concepts*, which are generic units of functionality. Framework-based applications are developed by writing *application code* that instantiates those concepts [4–6]. For example, the Eclipse framework offers concepts such as *viewers* and *editors*. Eclipses *Package Explorer* and *Java Editor* are instances of these concepts. Consequently, one of the most important parts of a framework for application developers is the *Application Programming Interface* (API) of that framework [7]. However, many of the existing frameworks often have complex and large APIs, and typically suffer from the lack of proper documentation [8]. To address these issues, developers usually try to investigate available sample applications to realize how to use a given API; this is what Gamma et al. [9] refer to it as the *"Monkey See/Monkey Do"* rule. Nevertheless, looking for sample applications over and over is an irritating job. In addition, users may fail to find their ideal example applications.

To tackle the above issues, a number of framework comprehension approaches have been proposed in literature. For instance,

* Corresponding author. Tel.: +982166166648; fax: +982166019246.
E-mail addresses: enoei@ce.sharif.edu (E. Noei), heydarnoori@sharif.edu (A. Heydarnoori).
URL: http://sharif.edu/~heydarnoori/ (A. Heydarnoori)

http://dx.doi.org/10.1016/j.infsof.2016.03.007
0950-5849/

Tran et al. [10] mine the structure and the contents of API documentation to find relevant methods. Cubranic et al. [11] propose an approach to recommend some artifacts that are relevant to a task that a developer is currently conducting. Exemplar [12] uses API calls executed by an application to search for applications. CodeGenie [13] allows the programmers to design test cases for a specific feature. Then, CodeGenie looks for a sample implementation based on the information in those test cases. Designing such test cases can be a tedious task when users are not much familiar with a framework.

The majority of existing framework comprehension approaches can only help programmers to get familiar with the usage of particular fine-grained API elements of the desired framework such as classes and methods. Nevertheless, none of them help novice developers to find suitable sample applications in the absence of appropriate documentation. According to the *"Monkey See/Monkey Do"* rule, in the case of lacking enough documentation and manuals, programmers try to have a look at available example applications to learn how to implement their desired framework-provided concepts.

With respect to above discussions, there can be the following issues in using application frameworks [14]: (i) large and complex APIs; (ii) lack of enough documentation; (iii) available documentation may be inaccurate or imprecise; (iv) making proper documentation is a hard and time-consuming task which prevents developers to create them adequately; (v) every single concept implemented in the example application is not necessarily described in the documentation; (vi) novice programmers are not familiar with the details of using a particular framework; and (vii) finding appropriate sample applications is a difficult task. To mitigate these problems, in this paper, we propose *EXAF* (*EXample Applications Finder*) to help programmers find appropriate example applications that implement their desired concepts on top of a particular framework.

In our approach, we suppose that a user is not an expert to use the framework [15]. Thus, the user would not know the exact name of the desired concept in the jargon of that particular framework. Consequently, we expand the user-provided keywords using the *Latent Semantic Indexing* (*LSI*) technique [16]. Next, we search for relevant sample applications in software projects hosting sites, such as SourceForge[1] and Google Code[2], with the help of the expanded keywords. For this purpose, we analyze different artifacts of available sample applications such as users' comments and bug reports. From the information extracted from these artifacts, we may probably find a number of sample applications that implement the desired concept. Finally, we rank the results and present them to the user. Our evaluations show that the precision of our approach is more than 79% that is more than the precision of available code search engines such as the ones provided by Google Code and SourceForge.

Our proposed approach in this article complements our earlier work on *FUDA* [4–6]. FUDA automatically generates *concept-implementation templates* from dynamic traces collected at runtime from sample applications. The FUDA's concept-implementation templates are Java pseudocodes that summarize necessary implementation steps required to implement a desired concept on top of a particular framework. Hence, the work presented in this article automates the FUDA's manual step of finding sample applications.

The contributions of this article include: (i) finding example applications of framework-provided concepts regardless of the quality and the availability of a framework's documentation; (ii) providing a solution for FUDA to find required example applications automatically; and (iii) applying the LSI information retrieval technique, and taking into account code comments and user reviews in the search process to get better results in terms of accuracy and precision compared to powerful general-purpose search engines like Google.

The remainder of this paper is organized as follows. Section 2 provides a motivating example of how EXAF works. Section 3 presents the details of EXAF. Next, Section 4 describes our implementations of EXAF. Afterwards, Section 5 discusses the evaluation method and the results of our evaluations. Section 6 has an overview of related work and compares them with our proposed technique. Finally, Section 7 concludes the paper.

2. Motivating example

Assume a programmer wants to implement a *context menu* on top of the *Java Swing* framework. Suppose this programmer is not an expert in Java Swing and thus, does not know the name of the desired concept in Java Swing. More specifically, the programmer is not aware of the fact that a "context menu" is referred to as a *pop-up menu* in Java Swing. Consequently, just looking for the term "context menu" reduces the chance of finding proper example applications in existing software projects repositories.

EXAF tackles the above issue via expanding the term "context menu" to other relevant terms, such as *pop-up menu* and *qmenu*. For this purpose, EXAF uses the pages from the Stack Overflow[3] website as the main resource for creating its corpus, and then applies the LSI information retrieval technique on it. This helps EXAF to expand the user-provided keywords into the domain of software engineering and programming. On the other hand, if EXAF had applied alternative approaches like using *WordNet* [17], which is a lexical database for the English language, it would had got irrelevant terms, such as *bill, dish*, and *card* for the term "context menu". This would have caused EXAF to generate inappropriate results.

EXAF not only uses the descriptions of applications, but also looks through the reviews to find proper sample applications that implement a desired concept on top of a particular framework. In the case that the name of the concept is not mentioned in the descriptions of applications, EXAF looks into the reviews. For instance, Fig. 1 shows a sample project in SourceForge that the term "context menu" has been mentioned in reviews, but not in that project's descriptions. This example illustrates that by not considering the reviews, we would have missed such sample applications. EXAF ranks the retrieved projects based on the expanded keywords and other factors, such as the title of the project (see Section 3.3). Thus, the programmer would get a ranked list of projects that implement a "context menu" in Java Swing.

To indicate how the ranking process of EXAF works, we use the following notation to show the sample projects retrieved by EXAF for the concept "context menu": $< t, d, <r> >$ in which t is the title of the project, d is the description of the project, and $< r >$ is the set of all reviews for that project. Now, consider the following two examples:

Example 1. Suppose the following two sample applications are retrieved:

1. $<$ *Awesome context menu: This project implements context menu with a very user-friendly interface,* $< \emptyset > >$
2. $<$ *Business management assistant: This program helps managers to come up with the difficulties of managing [...],* $<$ *I cannot add my account name, in the context menu there is an option [...]* $> >$

EXAF ranks the above two sample applications using a customized version of *BM25F* [18] (see Section 3.3 for details). This

[1] http://www.sourceforge.net
[2] http://code.google.com
[3] http://www.stackoverflow.com

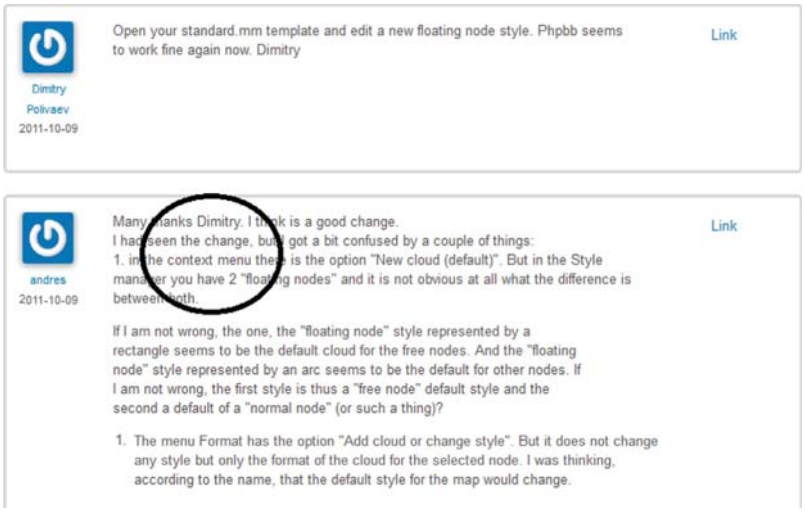


Fig. 1. A sample project in SourceForge. In users' review, the term *context menu* can be seen while nothing about it is mentioned in the project's description.

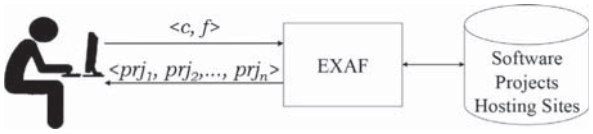


Fig. 2. The usage scenario of EXAF; $< c, f >$ is the input query in which c denotes the desired concept, and f is the name of the framework; $< prj_1, prj_2, ..., prj_n >$ is a ranked list of retrieved sample applications.

causes the first project to be returned first, and the second one comes last. This is because the "context menu" is mentioned both in the title and the description of the first project, but only in the reviews of the second project. In our customized version of BM25F, the title and the descriptions of projects are considered more important than that project's reviews [19].

Example 2. Now, assume the following two sample projects are returned:

1. $<$ *Business management assistant: This program helps managers to come up with the difficulties of managing [...],* $<$ *I cannot add my account name, in the context menu there is an option [...]* $> >$
2. $<$ *Word editor: This word editor allows users to edit text [...],* $<$ *Run time error fixed, Context menu appears not in the right place, Context menu does not have an option to copy the text* $> >$

EXAF ranks the above two projects by giving the second project a higher rank than the first one. This is because the context menu is mentioned more frequently in the reviews of the second project compared to the first one.

After getting the desired sample applications, the user can employ the FUDA [4–6] technique to get a summary of the concept-implementation steps and locate them in those example applications.

2.1. Usage scenario

To summarize this section, Fig. 2 illustrates an overview of EXAF's usage scenario. To find example applications that implement a concept c on top of a framework f, the user has to specify the arguments of the query which is in the form of a pair $< c, f >$. For instance, to search for sample applications that implement a "context menu" on top of the Java Swing framework, the query would be $<$ *context menu, java swing* $>$. The input query then gets expanded by EXAF into a set of queries. For instance, our example query would be expanded to $<$ $<$*context menu, java swing* $>$, $<$*popup menu, java swing* $>$, $<$*qmenu menu, java swing* $> >$. In response, the user gets a ranked list of sample applications that implement the desired concept on top of the specified framework in the form of a n-tuple ordered set $< prj_1, prj_2, ..., prj_n >$ such that n is the number of retrieved sample applications, and prj_i is the ith application. Thus, the prj_1 is the most relevant sample application, and prj_n is the least relevant one based on EXAF's ranking. The user can then download the *working* sample applications from the software projects hosting sites with the help of the URLs to those sample applications provided by EXAF. Next, the user can use the FUDA technique to locate the desired concept and get a summary of steps that have to be taken to implement that concept.

3. Proposed approach: EXAF

EXAF includes three main steps as illustrated in Fig. 3. First, the user provides a number of keywords that describe the concept of interest. EXAF then expands those keywords by applying the LSI technique. Second, EXAF looks for available projects in software projects hosting sites like Google Code and SourceForge to find example applications that may implement the user's desired concept. Third, found sample applications are ranked and presented to the user. In the following subsections, we describe the details of these steps.

3.1. Expanding the keywords

As Fig. 3 shows, EXAF expands the user-provided keywords using the LSI technique. This has two main advantages in *EXAF*. First, the user may not know the exact terms that express the desired concept since she/he may not be familiar enough with that particular framework. For instance, a user may use the term *pop-up menu* instead of the *context menu* to express the desired concept which is the "context menu" in this particular example. However, the term "pop-up menu" is meaningless in the jargon of the intended framework. Second, the sample applications in software projects hosting sites that we are looking for might have used different terms for the concept of interest in their artifacts like code descriptions, comments, and source codes. Consequently, the input query $< c, f >$ gets expanded to a set of queries $<$ $<c, f>$, $<e_1, f>$, $<e_1, f>$, ..., $<e_n, f>$ $>$ where c is the input concept, f shows the name of the desired framework, e_i is the ith expanded keyword, and n is the number of expanded keywords.

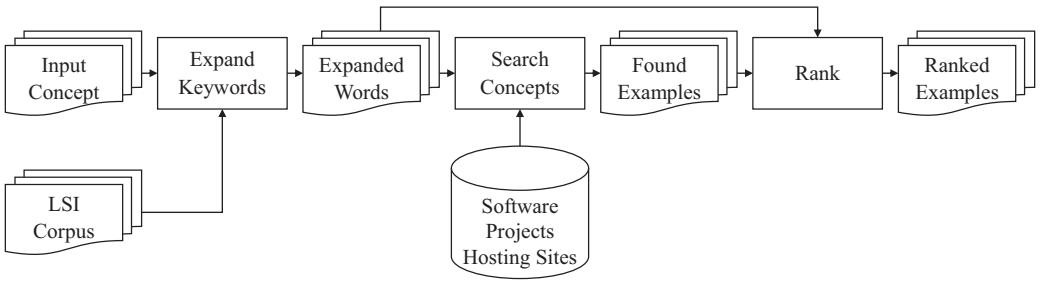


Fig. 3. The process of EXAF.

Similar to some earlier work in literature with respect to software engineering specific WordNets, like $WordSim_{DB}^{SE}$ [20] and *SE-WordSim* [21], EXAF also investigates online resources to expand the keywords. However, EXAF applies the *LSI* information retrieval technique [16] to expand the user-provided keywords. Information retrieval techniques are mostly based on processing external documents which may be costly in terms of gathering all the relevant keywords together. Nevertheless, our LSI corpus is based on the keywords extracted from the pages of Stack Overflow. More specifically, it only depends on the corpus that can be created without any manual efforts such as filtering out irrelevant concepts and keywords. The topics discussed in Stack Overflow are mainly related to computer science and software development. Stack Overflow is a Q&A website which features questions and answers on a broad range of topics in computer programming and information technology. In the following subsection, we introduce the LSI technique in more details and that how it is used in EXAF.

3.1.1. Latent semantic indexing (LSI)

LSI [16] is an approach for indexing data and retrieving information. LSI works based on the mathematical and statistical principles. The LSI approach builds a matrix, called *singular value decomposition* (*SVD*), to detect the relations between words in raw texts with no prior assumption on the semantic relations amongst them. Hence, the degree of the similarity of words can be computed. The raw text documents are called the *LSI corpus*. The LSI technique assumes that if the words go more often with each other, they may have the same meaning or have a tight-knit relationship with each other.

Fig. 4 shows the algorithm of LSI. As shown in Fig. 4, the first step of LSI computations is to create a large matrix, called the *occurrence matrix*, for comparing the words. Each row of this matrix belongs to a word, and each column belongs to a document. The similarity of the words can be computed by comparing the rows of this matrix. The occurrence matrix is a $m \times n$ matrix. As this matrix is a quiet large and usually sparse matrix, the LSI technique suggests to break it into three smaller matrices as follows:

$$A = T \times S \times D^{\mathrm{T}} \tag{1}$$

In Eq. (1), A denotes the occurrence matrix, T is called the *term-document matrix* (a $m \times r$ matrix), S is the SVD matrix, and D^{T} is the transpose of the *concept-documents matrix* (a $n \times r$ matrix). These three matrices will be reduced in dimension by applying some techniques like merging less related words together. Finally, the occurrence matrix will be rebuilt with very smaller matrices compared to the initial ones as follows:

$$A \approx A_K = T_K S_K {D_K}^{\mathrm{T}} \tag{2}$$

In Eq. (2), k demonstrates the new dimension of matrices after the reduction and compression process ($k \ll r$); T_K, S_K, and ${D_K}^{\mathrm{T}}$ respectively denote the term-document matrix, SVD matrix, and the transpose of the concept-documents matrix, after the size reduction. In EXAF, the user-provided keywords will be compared to all other words available in the occurrence matrix and the words that have a similarity higher than a particular threshold will be selected as the expanded keywords.

Fig. 5 shows a discussion about an issue related to the *Context Menu* concept in the Stack Overflow website. As it is shown in this figure, there are several keywords, such as *Pop-up Menu*, that are related to the *Context Menu* concept. So, by analyzing a number of pages like this, the keywords related to the concepts of interest could be detected. We used Stack Overflow to create our LSI corpus because of the following two main reasons:

- *The easiness of creating the corpus*: Stack Overflow is a popular Q&A website and thus, it includes many Q&A pages in which related keywords are kept together. Hence, it was straightforward to treat each Stack Overflow's Q&A page as one document and use those documents to create the corpus.
- *Unrelated keywords get eliminated automatically*: One of the difficulties of applying the LSI technique is the words that are not related to input keywords for the desired concept. However, since Stack Overflow features questions and answers on a range of topics in the domain of computer programming, the LSI corpus created from its pages would not propose out-of-domain keywords. For instance, words like *dish* and *restaurant* would never be expanded for the keyword *menu* for our *Context Menu* example since they are out of the domain of computer programming.

3.2. Searching software projects hosting sites for sample applications

After applying the LSI technique to expand the user-provided keywords that describe the concept of interest, we can search in software projects hosting sites like SourceForge to find example applications implementing that concept. Fig. 1 is a screenshot of a SourceForge page which shows comments for a particular project. As can be seen in this figure, the comments indicate that the *Context Menu* concept has been implemented in that project. However, there are no words related to the *Context Menu* within the descriptions of this project.

To address the above issue, EXAF expands user-provided keywords using the LSI technique and use them afterwards to find sample projects in SourceForge that implement the desired concept. To increase the precision of the search process, as shown in Fig. 6, EXAF conducts a two-step approach for finding relevant sample applications in SourceForge. In the first step, the keywords entered by the user will be examined independently from the expanded keywords. If the first step fails to find any sample applications, the second step will start. At the second step, the expanded keywords would be searched instead of the user-provided keywords to find relevant sample applications implementing the desired concept. In other words, since expanded keywords are somehow related to the original user-provided keywords based on the LSI techniques, the sample applications mined using them would

```
function EXPAND(kw: keyword, docs: Array of corpus documents, dim: dimension, sim: similarity): Array of expanded words
    expandedWords: array of words
    A : Matrix                                                    ▷ array of words
    for all d: document in docs do
        for all w: word in d do
            if d ∈ A then
                increase the frequency of w in d
            else
                create a row and set frequency of w in d to 1
            end if
        end for
    end for
    kv ← vector(kw)
    S, T, D^T, S_k, T_k, D_k^T, A_k: Matrix
    S, T, D^T ← A
    S_k, T_k, D_k^T ← reduceRank(S, T, D^T)
    A_k ← S_k × T_k times D_k^T
    for all w: word in A_k do
        if (Cos(kv, vector(w)) ≥ s then
            expandedWords ← expandedWords ∪ w
        end if
    end for
    return expandedWords
end function
```

Fig. 4. The LSI algorithm.

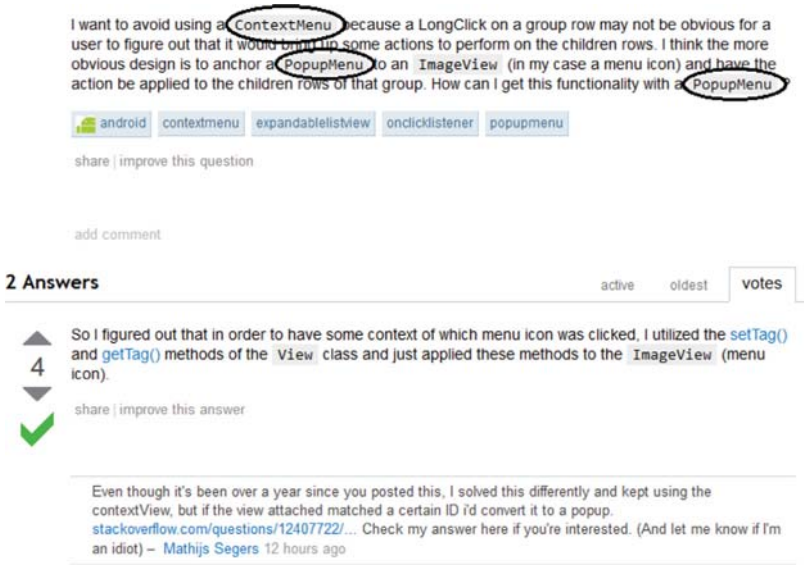


Fig. 5. A discussion related to the concept "context menu" in the Stack Overflow website. However, the keyword *popup menu* can be observed as well.

also be somehow relevant to the desired concept. For instance, the expanded keywords could be the synonyms of the user-provided keywords or can be those keywords that have a tight-knit relationship with them. As an example, the keywords *Context Menu* and *Pop-up Menu* could be used interchangeably.

Another advantage of the two-step approach is that it can increase the precision of the search process (see evaluations in Section 5). In the first step, the exact keywords entered by the user are searched. As the keywords provided by the user might specify the desired concept more accurately than those which are expanded, the sample applications found in the first step could be more relevant than those found in the second step (see evaluations in Section 5). However, the two-step approach using the LSI technique increases the chance of finding relevant applications. In other words, as discussed in the above example (cf. Fig. 1), only using the user-provided keywords does not return any relevant results. However, after expanding the user-provided keywords, the desired sample applications might be found. More specifically, after expanding the user-provided keywords, EXAF finds more relevant applications because of two primary reasons: (i) according to our observations, users do not necessarily use the same terms to explain a particular concept [15,22,23]; and (ii) novice users often

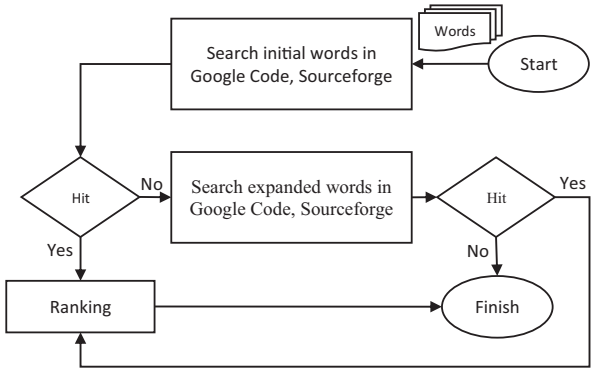


Fig. 6. The search process.

do not know much about the exact names of concepts in the jargon of a particular framework [23,24].

3.3. Ranking retrieved sample applications

After searching software projects hosting sites for relevant example applications, the next step of EXAF is to rank found sample applications based on their relevancy. For this purpose, we respectively take the following factors into account: (i) the title of the project; (ii) the description of the project; (iii) the tags of the project; and (iv) the comments, feedbacks, and the reviews of the project. The rationale behind this prioritization is that (i) as stated by Edmundson [19], the title typically circumscribes the subject matter of the text; and (ii) the developers or the owners of the projects usually write the descriptions of projects and assign tags to them, where the reviews can be written by third-party users [25]. Hence, we conjecture that the descriptions of projects would be more accurate than the reviews.

To rank the retrieved sample applications, we customize *BM25F* [18] with *length normalization* with respect to the above factors. BM25F is a standard retrieval function that ranks structured documents based on the relative proximity of query terms. EXAF also filters out the results that are not relevant to the desired framework by investigating the above factors of sample applications.

$$score(a, Q) = \sum_{q\ in\ Q} log\frac{N - n(q) + 0.5}{n(q) + 0.5} \times \frac{w(a, q)}{k_1 + w(a, q)}. \quad (3)$$

Eq. (3) shows the scoring formula to rank a sample application a by a set of keywords Q (q_1, q_2, ..., q_n). In Eq. (3), N denotes the total number of results; $n(q)$ shows the total number of results containing the keyword q; $w(a, q)$ denotes the weight of keyword q; k_1 is a free parameter that controls the scale of that keyword's frequency; and 0.5 is a constant to deal with the cases that $n(q) = 0$.

The weight of keywords can be calculated by Eq. (4) in which p_s represents the importance of each part, including subjects, tags, and reviews. We assigned 4 to subjects, 2 to descriptions and tags, and 1 to reviews, based on our analyses for the best results. $f^a_{q,s}$ shows the frequency of keyword q that occurs in the part s of application a; l_s is the length of part s; avl_s is the average length of part s in all of the sample applications found by EXAF; and b_s is a free parameter to control the document length. Based on our investigations, similar to Shi et al. [26], we finally came up with $k_1 = 2$ and $b_s = 0.75$ to get the best results in terms of accuracy. This ranking function is also implemented in the *Apache Lucene*[4] framework which is used in our EXAF implementations (see Section 4):

$$w(a, q) = \sum_{s\ in\ a} \frac{p_s \times f^a_{q,s}}{(1 - b_s) + b_s \times \frac{l_s}{avl_s}}. \quad (4)$$

4. The EXAF implementation

We implemented EXAF as an Eclipse plug-in. This implementation was then used in our evaluations of EXAF as will be discussed in Section 5. The implementation of EXAF includes two main components that are described in the following: (i) the LSI component, and (ii) the Search Engine component.

4.1. The LSI component

To implement the EXAF's LSI component, we have to take care of the following two issues: (i) building the LSI's corpus, and (ii) performing the LSI computations. To build our corpus, we analyzed more than 6,000 pages of Stack Overflow. There are five important sections in each Stack Overflow page: (i) the title, (ii) a question, (iii) the responses to that question, (iv) the code snippets, and (v) the tags. We analyzed each section by extracting the keywords in that section and putting aside the irrelevant words such as external-links and stop-words [28]. To conduct our LSI computations, we got advantage of the *S-space* package [29].

One of the challenges of the LSI technique is to find the best dimension to reduce the size of the term-document matrix. The term-document is a large and sparse matrix which has to be cut in dimension to make the computations more efficient. The smaller dimensions make the calculations faster and more practical while the bigger dimensions make the results more accurate. As pointed out in [30], a dimension around 300 usually gives the best results, especially when the number of documents is hundreds or thousands, and a dimension around 400 is suitable for millions of documents. However, another work [16] shows that the dimension between 50 and 1000 has the best results depending on the contents of the documents and their size. Therefore, we tested and analyzed the dimensions of 50, 100, 200, 300, 400, and 800. We observed that the best results are based on a dimension around 300.

Another challenge is the degree of similarity. As the words are compared to each other using the Cosines similarity, a threshold must be set to get the best results. We tested and investigated several sample concepts to specify an appropriate threshold. Finally, the similarity threshold of 0.7 with the dimension of 300, the similarity of 0.57 with the dimension of 400, and the similarity threshold of 0.52 with the dimension of 800 gave us the best results in our evaluations (see Section 5). Based on our experiments and with respect to Bradford's statement [30] that the dimension of 300 is mostly used by researchers for the best results, we chose the similarity threshold of 0.7 with the dimension of 300 for EXAF to have a more efficient solution.

4.2. The search engine component

To have a fast and efficient implementation of EXAF's Search Engine component, we used both *Apache Solr*[5] [31] and *Apache Nutch*[6]. Apache Solr is an open source enterprise search platform built on top of the Apache Lucene which is a full-featured text search engine library written in Java. Apache Nutch is also an open source web search engine based on Lucene. We customized these

[4] http://lucene.apache.org/

[5] http://lucene.apache.org/solr/

[6] http://nutch.apache.org/

Apache open source engines in such a way that they satisfy EXAF's requirements. For instance, before indexing the words of a page, we conducted a *pre-indexing* step to avoid indexing all the contents of that page except the required ones, such as descriptions and comments. Moreover, we tokenized on white-spaces and removed stop-words. After that, we handled special cases, such as dashes. Then, we lowercased all the terms. Finally, we stemmed the documents using the *Porter English* algorithm [32].

Apache Solr's architecture includes three layers: (i) the interaction, (ii) the Solr core, and (iii) the storage [33]. We used the *SolrJ*[7] client to access Solr. The Solr application layer is mainly responsible for handling the relation between the Solr and the external elements, such as handling the requests. There are also several processing units such as de-duplication and language detection units. Solr also uses *Apache Tika*[8] to detect and extract metadata and text from different file types. The Apache Lucene is responsible for indexing data and searching among the indexed documents. Finally, the indexed documents will be saved in the index storage, and any information about the schema and also the metadata will be saved in a database.

We crawled and indexed the SourceForge website with a *width* of 500 and a *depth* of 5000. Depth shows the number of addresses to be crawled, starting from the initial seed. Width shows the number of pages to be added to the address queue. To test our search engine, we crawled and retrieved the information of 756,730 pages that belong to 95,172 unique projects. In addition to the SourceForge, to test our search engine on other software projects hosting sites, we also crawled Google Code, but in a smaller scale. We crawled and indexed the Google Code with a width of 300 and a depth of 500, and we retrieved 12,838 pages that belong to 1632 unique projects on the Google Code.

5. Evaluations

This section presents the evaluations of EXAF, including the evaluation objectives, the evaluation setup, and the evaluation results.

5.1. Evaluations objectives

In our evaluations, we are in favor of answering the following research questions:

1. Can EXAF retrieve proper sample applications for desired concepts from software projects hosting sites like SourceForge and Google Code?
2. Can EXAF produce more relevant results compared to other search methods like the search engines used by SourceForge and Google Code, or the general web search?

5.2. Evaluations setup

To answer the research questions mentioned in Section 5.1, we followed the following steps to perform the evaluations of EXAF:

Selection of frameworks. We evaluate EXAF on top of the *Microsoft .NET*, *Qt*, and *Java Swing* frameworks. The reasons of choosing these three frameworks are their complexity, popularity, and applicability [34]. Microsoft .NET is an application framework that runs on Microsoft Windows [35]. The Qt framework is a cross-platform framework that is used for developing applications that can be run on various software and hardware platforms [36]. Finally, the Java Swing is a popular GUI framework for Java [37].

[7] https://wiki.apache.org/solr/Solrj/
[8] https://tika.apache.org/

Table 1
Selected concepts on top of the Microsoft .Net, Qt, and Java Swing frameworks.

Concept	Description
Context menu	A menu in a graphical user interface that appears upon user interaction, such as a right-click mouse operation.
Table viewer	An object for demonstrating tables.
Tree viewer	An object for demonstrating trees.
Timer	Keeps track of how much time has been spent.
Database connection	An object for connecting to Database Management Systems.
Card layout	A layout manager for a container.
Shape	A graphical item with external boundary and outline.
Navigate	Access different objects.
Array	A data structure consisting of a collection of elements.
Moving shape	A shape that its position can be changed.
Generic interface	An object that provides common functionality across families of generic types.
Priority queue	A queue with some priority rules.
Combo box	A user input device in which the user can select an option from a drop-down list or type in a value into a text box.
Vector	A data structure for storing quantity with magnitude and direction.
Dialog	An interaction tool.
TCP connection	An object for that provides a network connection using TCP protocol.
Label	An object for showing unchangeable texts.
Progress bar	An indicator that shows the current status of a task.
Radio button	A circle representing choices in a common options list form in a graphical user interface.
Hashmap	A data structure of hash.
Stack	A data structure with first in, last out policy.
Graph	A graphical representation object.
Thread	A thread of execution.

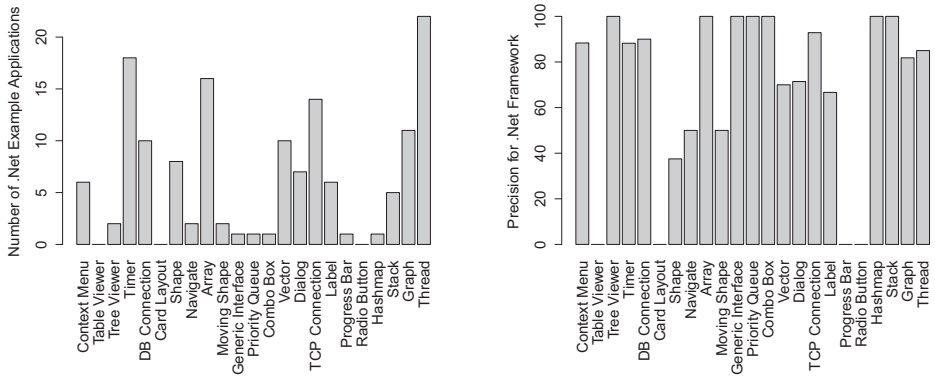
Selection of concepts. We investigate the results of EXAF in finding example applications for the 24 concepts that are listed in Table 1 on top of the .NET, Qt, and Java Swing frameworks. Table 1 provides a brief description of each concept as well. We sampled the concepts from the developer forums of the respective frameworks to answer real development issues.

Creating the queries and performing the search. For each of the 24 concepts listed in Table 1, first we created a query statement in the following way: a pair $< c, f >$ is used to formulate the input query where c is the desired concept, and f is the name of the desired framework. Next, we used our implementation of EXAF (see Section 4) to find sample applications for each of those concepts on top of the Microsoft .NET, Qt, and Java Swing frameworks. The output of EXAF is a ranked list of retrieved sample applications ordered by their relevancy to the input query.

5.3. Evaluations results

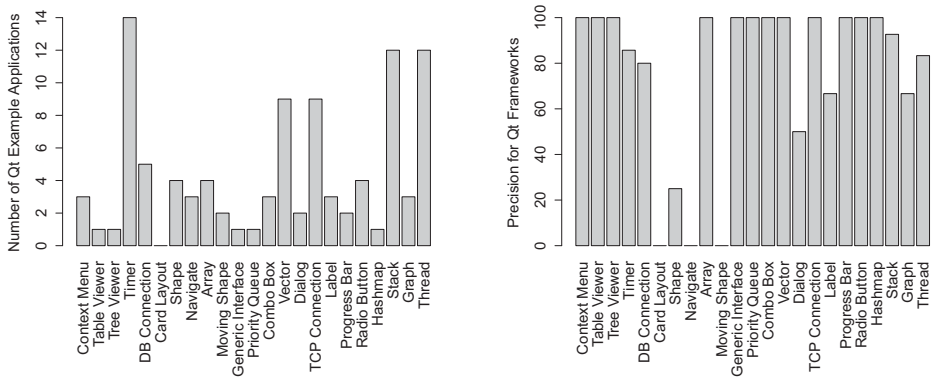
This section presents the results of our evaluations of EXAF.

The number of results. Fig. 7a shows the number of retrieved example applications from the SourceForge for each concept presented in Table 1 for the .Net framework. Similarly, Fig. 8a and Fig. 9a respectively illustrates the number of retrieved example applications from the SourceForge for each sample concept on top of the Qt and Java Swing frameworks. As indicated in these figures, EXAF found a total of 139, 99, and 77 example applications for our sample concepts on top of the .Net, Qt, and Java Swing frameworks respectively.



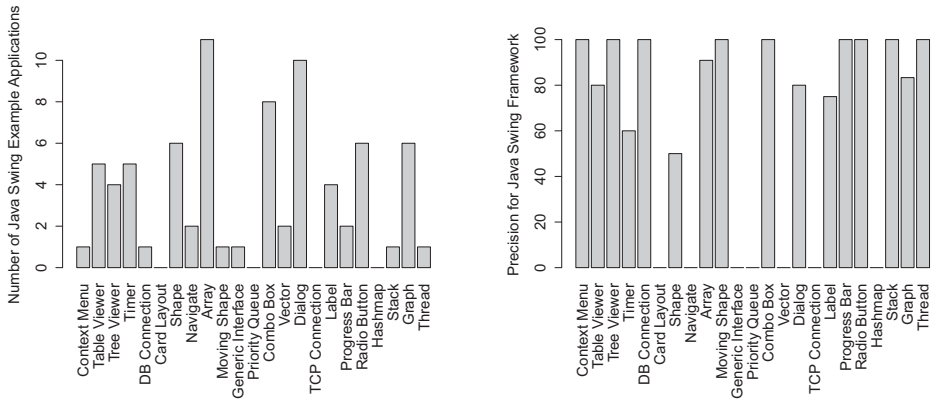
(a) The number of example applications (b) The precision for different concepts

Fig. 7. The precision and the number of retrieved example applications from the SourceForge for the Microsoft .Net framework.



(a) The number of example applications (b) The precision for different concepts

Fig. 8. The precision and the number of retrieved example applications from the SourceForge for different concepts for the Qt framework.



(a) The number of example applications (b) The precision for different concepts

Fig. 9. The precision and the number of retrieved example applications from the SourceForge for different concepts for the Java Swing framework.

The precision of results. As mentioned before, Figs. 7, 8 and 9 show that EXAF was able to find an overall of 139, 99, and 77 sample applications for the concepts listed in Table 1 on top of the .Net, Qt, and Java Swing frameworks respectively. For these frameworks, the precisions of results were 81.71%, 82.83%, and 79.22% respectively. In particular, as illustrated in Fig. 7b, EXAF achieves a precision of 100% in finding sample applications for 7, 12, and 9 of the concepts listed in Table 1 on top of the .Net, Qt, and Java Swing frameworks respectively. These promising results are mainly because of the fact that EXAF takes into account the comments and reviews as a key in finding proper example applications, and also benefits from the LSI information retrieval technique.

A case study on the Google code. What presented so far in this section, shows the results of our evaluations of applying EXAF on the SourceForge. To have a more comprehensive view of how effective the EXAF is in practice, we applied EXAF on the Google Code with a smaller number of crawled projects (i.e., 1,632 projects). To save

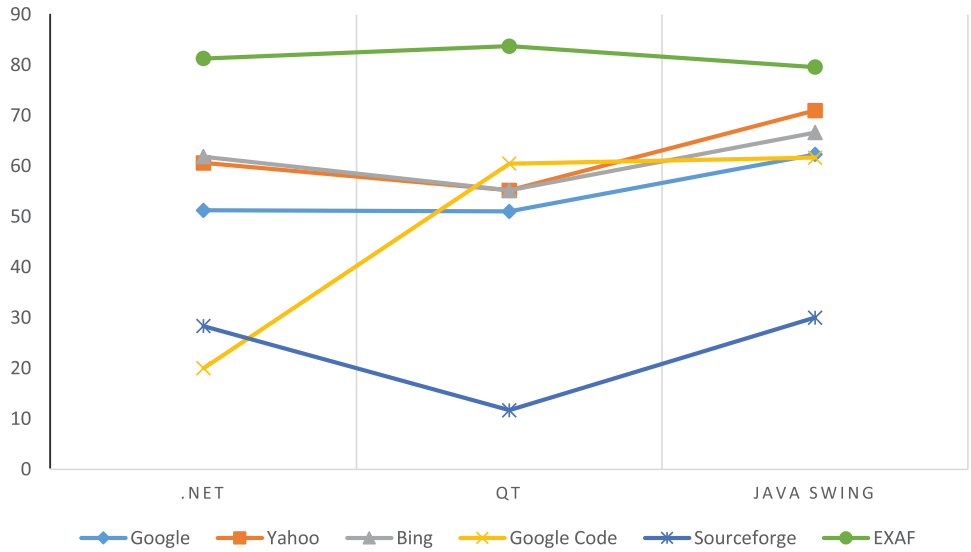


Fig. 10. Comparing the average precision of results of EXAF with the average precision of results of other search engines in finding sample applications for the concepts listed in Table 1.

space, in the following, we briefly provide the final results of this experiment as well.

After following the same steps of our experiment with the SourceForge, we achieved the following results: EXAF found 29 example applications for the concepts listed in Table 1 on top of the .Net framework with a precision of 79.34%, 24 example applications on top of the Qt framework with a precision of 87.5%, and 21 example applications on top of the Java Swing framework with a precision of 81%.

Comparing EXAF with other search engines. In this section, we compare the results of applying EXAF to SourceForge and Google Code, with the results of applying general-purpose search engines of Google, Yahoo, and Bing to them. We also compare the results of EXAF with the results of search engines incorporated in SourceForge and Google Code themselves. As the number of retrieved results using these engines is often vast and numerous, and based on the fact that most of the users are interested in the first few results [38], we calculated the precision for the first 10 results of each of these engines. To perform the search for each concept, we created our query using the name of that concept plus the name of the desired framework. Additionally, for general-purpose search engines of Google, Yahoo, and Bing, we limited the results once to those found in the Google Code and once, to those found in the SourceForge.

Fig. 10 presents the results of comparing EXAF with other search engines. As illustrated in this figure, EXAF works better than the above five search engines in terms of the precision of results. We noticed in our evaluations that the main reasons for these improvements are: (i) general-purpose search engines are not particularly provided to find sample applications for framework-provided concepts; (ii) in many cases, they refer to available documentation in sample projects while most of the times, the desired concepts are not necessarily discussed in them; and (iii) different results of a query may include different parts of the same project which can hinder the precision of results.

5.4. Threats to validity

In the following, we discuss the threats that may influence the validity of the experiment results, presented in the preceding sections.

5.4.1. Internal validity

Internal validity relates to the extent to which the design and analysis may have been compromised by the existence of confounding variables and other unexpected sources of bias [39].

One of the threats to internal validity concerns our selection of artifacts that we take into account in EXAF to look for relevant sample applications. For instance, we could have asked users to provide us the API elements that they think might be relevant to their desired concepts. Taking into account more artifacts may improve the search results. Nevertheless, as mentioned before, we assume that the user is not familiar enough with the target framework, and that there are not enough documentation and guides available describing how to implement a desired concept on top of that framework. Hence, to make EXAF more applicable and simpler, we prefer not to use other artifacts.

Another threat to internal validity relates to our selection of software projects hosting sites, i.e., SourceForge and Google Code, that can influence the results of our evaluations. We chose these two sites because of their number of projects and popularity. However, there are many other software projects repositories, such as GitHub, that could have been considered in EXAF too.

5.4.2. External validity

External validity relates to the extent to which the research questions capture the objectives of the research and the extent to which any conclusions can be generalized [39].

One of the threats to external validity is that the frameworks we used in our evaluations are not representatives of those used in realistic development. However, we addressed this threat by selecting three large and complex frameworks with various properties, i.e., .Net, Qt, and Java Swing, that are widely used in practice. However, it is still useful to perform the evaluations with other frameworks as well.

Another threat to external validity is that the concepts selected for our evaluations might not be representatives of real-world problems. We addressed this threat by selecting real-world concepts from developer forums.

As pointed out earlier in Sections 1 and 2, EXAF was originally developed as a complement of our earlier work on FUDA. More specifically, EXAF automates the FUDA's manual step of finding sample applications, and thus, users can benefit from both of these tools together to learn how to implement a particular concept

on top of a desired *object-oriented* application framework. Consequently, we introduced EXAF in this article as a search engine for sample applications of object-oriented framework-provided concepts. However, if a user wants to benefit from EXAF independently from FUDA, it works independently of any programming paradigm since it textually searches through various artifacts of a software project like user reviews and bug reports.

In the descriptions of most of the applications available in software projects repositories like SourceForge and Google Code, there is typically no description about what concepts they implement. Therefore, to find out whether an application implements a particular concept or not, EXAF searches through the comments, feedbacks, reviews, feature requests, and bug reports of that application as well. Therefore, EXAF might not be that helpful if a software projects repository does not include these artifacts about an application. Moreover, if a software projects repository which is used with EXAF does not have any sample applications implementing a desired concept, EXAF would not return any results.

5.4.3. Construct validity

The test of construct validity questions whether the theoretical constructs are interpreted and measured correctly [39]. In our evaluations, the main threat to construct validity is related to measuring the precision of results. In practice, only the actual user can state if the retrieved sample application is useful. This threat is minimized by inspecting the retrieved sample applications and calculating the precisions by two expert developers.

Another threat to construct validity is that we do not measure the *recall* of results which is the fraction of relevant instances that are retrieved. The reason for this is that there are a large number of applications in Google Code and SourceForge. Therefore, as we do not know the exact number of relevant sample applications in them, we just computed the precision of results. Furthermore, typically, one or two sample applications would suffice for developers to learn how to implement a desired concept, and thus, it is not required to retrieve all relevant sample applications from those sites.

5.4.4. Reliability

To implement EXAF, we customized the Apache powerful set of open source tools. We built our LSI corpus using the Stack Overflow discussions. We also conducted our experiments on the SourceForge and Google Code which are publicly available. Hence, all the resources we used in our evaluations are available online. Consequently, it should be possible to replicate the evaluations.

6. Related work

This section provides an overview of related work in the areas of (i) recommendation systems, (ii) code search engines, and (iii) general-purpose search engines.

6.1. Recommendation systems

Recommendation systems help developers during programming tasks at hand via recommending relevant items from a repository of programming artifacts like code snippets, discussions, documentation, and so on [40]. There are two main approaches that are commonly used by available recommendation systems: code based, and non-code based ones [41]. Code based recommendation systems make their recommendations with respect to the source code (e.g., API calls [42,43]). On the other hand, non-code based recommendation systems consider other artifacts, such as textual descriptions [44].

There is a large body of work that statically mine the source codes of existing example applications of a particular framework to recommend code snippets and usage rules of that framework's API. For instance, *Strathcona* [45], *XSnippet* [46], *FrUiT* [42], *MAPO* [43], *ParseWeb* [47], *Spotweb* [48], *EasySearch* [10], *CodeBroker* [49,50], and *Hipikat* [11] are examples of this category of approaches.

Both Strathcona and XSnippet are context-sensitive code assistants in which with respect to the programming task at hand, relevant code snippets from a repository of sample applications are recommended to the programmer. Similarly, FrUiT mines frequent API usage patterns in the form of *association rules* (e.g., *CallMethodA*⇒*CallMethodB*) to suggest relevant implementation steps. MAPO searches open source repositories using a user-defined query characterizing an API by a method, class, or package. It then applies data mining techniques to extract patterns of sequential method calls. PARSEWeb mines for a sequence of calls that transform an object of type τ_{in} into another object of type τ_{out}. SpotWeb mines sample applications to determine the *hot-spots* and *cold-spots* of a framework API. Hot-spots are defined as frequently used API methods and classes, but the cold-spots are API methods and classes that are rarely used in client applications. Thung et al. propose an automated approach in [44] that takes as input a textual description of a feature request. It then recommends methods in library APIs that developers can use to implement that feature. EasySearch is an approach that combines keyword-based and semantic-based searches to find relevant API functions. For this purpose, it mines the structure and contents of API documentation. CodeBroker automatically recommends program components for reuse with respect to the programming task at hand and the background knowledge of the developer. Finally, Hipikat intends to help newcomers to an open-source project become productive faster. To this end, it forms an implicit group memory from the information stored in a project's archives. It then recommends artifacts from the archives that are relevant to a task which that newcomer is trying to perform.

Recommendation systems mainly depend on the knowledge of users about the frameworks APIs and the availability of documents and guides. Lack of documentation and the low level of knowledge of developers are the main challenges for these works. Nevertheless, EXAF reduces the risk of lack of knowledge by expanding the keywords and it is independent from any documentation. Moreover, EXAF finds a complete sample program for the desired concept while these approaches recommend fine-grained API elements. Moreover, our proposed approach can significantly help our earlier work on FUDA. FUDA is a semi-automated technique for automatic extraction of *concept implementation templates* from traces of sample applications collected at runtime while invoking the desired concept. A concept implementation template is a code snippet that summarizes the implementation steps that are necessary to instantiate that concept. Thus, developers can use EXAF to find relevant sample applications implementing a desired concept. Then, they can apply the FUDA technique on those sample applications to generate a code snippet that implements that particular concept.

6.2. Code search engines

There are a number of search engines that are developed to particularly search for desired source codes. Examples of this category of approaches include *Exemplar* [12], *Assieme* [51], *CodeGenie* [13], *XFinder* [52], *SNIFF* [53], *S*^{6} [54], *MUSE* [55], *Mica* [56], and Satsy [57].

Exemplar is a search engine that combines a natural language query from the user and the API calls executed by an application to search for relevant applications that implement a desired concept. However, unlike EXAF that applies the LSI technique to extend the

user-provided keywords, and also searches in various artifacts of a project (e.g., code comments), Exemplar only takes into account the API calls made by that project.

Assieme is a special purpose Web search engine with which users can search for specific API elements to get more information about them or to get sample code snippets about the usages of those API elements. Therefore, unlike EXAF that looks for sample applications implementing coarse-grained concepts, Assieme looks for sample code snippets of fine-grained API elements.

With CodeGenie, programmers first design test cases for a feature of interest. Next, CodeGenie automatically searches for a sample implementation based on the information available in those test cases. However, EXAF does not need developers to design such test cases. Furthermore, designing these test cases can be a challenging task, particularly when the application for which we want to design those test cases is not at our hand.

Given a concept implementation template written in *Mismar* [58], XFinder looks for instances of this template in its code base. Mismar is a concept-oriented documentation toolset that focuses on code artifacts and their relationships. Unlike this approach, EXAF does not need high-level documentation of the concept that a developer is looking for its sample applications.

SNIFF uses the documentation of the framework methods to add plain English annotations to undocumented methods in example applications of that framework. The annotated applications are then indexed for the purpose of free-form query search. However, unlike EXAF, this technique would not work when the documentation of the framework is not available.

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, not complete applications. Like EXAF, S^6 returns source code, however, it requires additional low-level details from the user, such as data types of test cases.

MUSE [55] parses the source code of the projects, and employs static slicing and clone detection to find example applications. However, EXAF uses textual artifacts, such as project descriptions, comments, and reviews to find proper sample applications.

Given a description of a desired functionality, Mica helps programmers find the right API classes and methods. For this purpose, Mica uses the Google Web APIs to find relevant pages, and then analyzes the content of those pages to extract the most relevant programming terms and to classify the type of each result. Mica also helps developers find examples when they already know which methods to use. Hence, unlike EXAF that searches in code repositories, Mica benefits from the Google general-purpose search engine and performs a general Web search.

In Satsy, programmers use an input/output query model to specify what behavior they want instead of how it may be implemented. Satsy includes a code repository in which programs are encoded as constraints, and applies an *SMT solver* to find encoded programs that match the input/output query. Satsy returns a list of source code snippets that match the specification. Therefore, in contrast to EXAF which looks for sample implementations of a desired concept, Satsy focuses on the behavior of programs regardless of how they implement the functionalities.

6.3. General-purpose search engines

General-purpose search engines are designed to search for information on the Internet (world wide web). Examples of this category of approaches include *Google*[9], *Bing*[10], and *Yahoo*[11].

General-purpose search engines are not particularly developed to search for relevant sample applications. Therefore, they not only search the source codes of sample applications available online, but also they search many other artifacts, like photos, raw texts, hypertexts, and books. Therefore, their suggested results cover a wide range of irrelevant artifacts. Hence, for finding relevant sample applications, users need to filter out irrelevant results manually which can be a tedious and time-consuming task. Nevertheless, EXAF makes it very easy for the users by suggesting only the sample applications that implement their required concepts.

7. Conclusions

Object-oriented application frameworks enable the reuse of design and code, and thus, make developing new applications simpler while improving their maintainability. Framework-based applications are developed by writing application code that instantiates framework-provided concepts. However, the APIs of modern frameworks are often large and complex, and suffer from the lack of proper documentation. To address these issues, programmers usually use existing sample applications as a guide to learn how to implement a particular concept. However, finding proper sample applications can be a cumbersome task. To address this difficulty, in this article, we introduced *EXAF*, that looks for relevant sample applications that implement a desired concept in software projects hosting sites like SourceForge and Google Code.

In EXAF, developers describe their desired concept in natural languages by a number of keywords. EXAF then expands those keywords by applying the LSI information retrieval technique. It then searches through the descriptions, bug reports, comments, reviews, feedbacks, and other artifacts of applications in software projects repositories to find example applications implementing the desired concept. EXAF ranks the results next with respect to their relevancy and presents them to the user.

We implemented EXAF as an Eclipse plugin and evaluated it with 24 real-world concepts on top of the .Net, Qt, and Java Swing frameworks. We noticed that EXAF has a precision of more than 79% and performs better than general-purpose search engines and also the code search engines integrated with SourceForge and Google Code.

In future, we plan to evaluate EXAF further with more frameworks and concepts. Moreover, we want to make EXAF publicly available as a website. In addition, it would be interesting to develop a tool that integrates both approaches of FUDA and EXAF to further help developers in getting concept-implementation templates automatically.

References

[1] W.B. Frakes, K. Kang, Software reuse research: status and future, IEEE Trans. Softw. Eng. 31 (7) (2005) 529–536.
[2] R. Keswani, S. Joshi, A. Jatain, Software reuse in practice, in: Proceedings of the 4th International Conference on Advanced Computing & Communication Technologies, IEEE, 2014, pp. 159–162.
[3] T. Scheller, E. Kühn, Automated measurement of API usability: the API concepts framework, Inf. Softw. Technol. 61 (5) (2015) 145–162.
[4] A. Heydarnoori, Supporting Framework Use via Automatically Extracted Concept-Implementation Templates, University of Waterloo, Canada, 2009 (Ph.D. thesis).
[5] A. Heydarnoori, K. Czarnecki, W. Binder, T.T. Bartolomei, Two studies of framework-usage templates extracted from dynamic traces, IEEE Trans. Softw. Eng. 38 (6) (2012) 1464–1487.
[6] A. Heydarnoori, K. Czarnecki, T.T. Bartolomei, Supporting framework use via automatically extracted concept-implementation templates, in: Proceedings of the 23rd European Conference on Object-Oriented Programming, in: vol. 5653 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, Germany, 2009, pp. 344–368.
[7] J.L.C. Izquierdo, F. Jouault, J. Cabot, J.G. Molina, API2MoL: automating the building of bridges between APIs and model-driven engineering, Inf. Softw. Technol. 54 (3) (2012) 257–273.

[9] http://www.google.com
[10] http://www.bing.com
[11] http://www.yahoo.com

[8] U. Dekel, J.D. Herbsleb, Improving API documentation usability with knowledge pushing, in: Proceedings of the 31st International Conference on Software Engineering, IEEE, 2009, pp. 320–330.

[9] E. Gamma, K. Beck, Contributing to Eclipse: Principles, Patterns, and Plug-ins, Addison-Wesley Professional, Redwood City, CA, USA, 2004.

[10] D.H. Tran, H.P. Nguyen, D.H. Le, EasySearch: finding relevant functions based on API documentation, in: Knowledge and Systems Engineering, Springer International Publishing, Switzerland, 2015, pp. 143–154.

[11] D. Čubranic, G.C. Murphy, Hipikat: recommending pertinent software development artifacts, in: Proceedings of the 25th International Conference on Software Engineering, IEEE, 2003, pp. 408–418.

[12] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, C. Cumby, A search engine for finding highly relevant applications, in: Proceedings of the 32nd International Conference on Software Engineering, 1, IEEE/ACM, 2010, pp. 475–484.

[13] O.A. Lazzarini Lemos, S.K. Bajracharya, J. Ossher, CodeGenie: a tool for test–driven source code search, in: Companion to the 22nd Conference on Object-oriented Programming Systems and Applications, ACM, 2007, pp. 917–918.

[14] M.P. Robillard, What makes APIs hard to learn? answers from developers, IEEE Softw. 26 (6) (2009) 26–34.

[15] E. Soloway, J.C. Spohrer, Studying the Novice Programmer, Psychology Press, Hillsdale, NJ, USA, 2013.

[16] T. Hofmann, Probabilistic latent semantic indexing, in: Proceedings of the 22nd Annual International Conference on Research and Development in Information Retrieval, ACM, 1999, pp. 50–57.

[17] C. Fellbaum, WordNet, 1998.Wiley Online Library.

[18] S.E. Robertson, S. Walker, M. Beaulieu, P. Willett, Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive track, Nist Spec. Publ. SP (1999) 253–264.

[19] H.P. Edmundson, New methods in automatic extracting, J. ACM (JACM) 16 (2) (1969) 264–285.

[20] Y. Tian, D. Lo, J. Lawall, Automated construction of a software-specific word similarity database, in: Proceedings of the 2014 Conference on Software Maintenance, Reengineering and Reverse Engineering, IEEE, 2014a, pp. 44–53.

[21] Y. Tian, D. Lo, J. Lawall, Sewordsim: software-specific word similarity database, in: Companion Proceedings of the 36th International Conference on Software Engineering, ACM, 2014b, pp. 568–571.

[22] K. Taneja, D. Dig, T. Xie, Automated detection of api refactorings in libraries, in: Proceedings of the 22nd IEEE/ACM international Conference on Automated Software Engineering, ACM, 2007, pp. 377–380.

[23] J. Sheard, A. Carbone, R. Lister, B. Simon, E. Thompson, J.L. Whalley, Going SOLO to assess novice programmers, in: ACM SIGCSE Bulletin, 40, ACM, 2008, pp. 209–213.

[24] C. Kelleher, R. Pausch, Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers, ACM Comput. Surv. (CSUR) 37 (2) (2005) 83–137.

[25] M. Pedroni, T. Bay, M. Oriol, A. Pedroni, Open source projects in programming courses, ACM SIGCSE Bull. 39 (1) (2007) 454–458.

[26] Z. Shi, J. Keung, Q. Song, An empirical study of BM25 and BM25F based feature location techniques, in: Proceedings of the International Workshop on Innovative Software Development Methodologies and Practices, ACM, 2014, pp. 106–114.

[27] EXAF, 2016. http://noei.bitbucket.org/EXAF/.

[28] W.J. Wilbur, K. Sirotkin,The automatic identification of stop words, J. Inf. Sci. 18 (1) (1992) 45–55.

[29] D. Jurgens, K. Stevens, The s-space package: an open source package for word space models, in: Proceedings of the ACL 2010 System Demonstrations, Association for Computational Linguistics, 2010, pp. 30–35.

[30] R.B. Bradford, An empirical study of required dimensionality for large-scale latent semantic indexing applications, in: Proceedings of the 17th Conference on Information and Knowledge Management, ACM, 2008, pp. 153–162.

[31] Z. Laliwala, A. Shaikh, Web Crawling and Data Mining with Apache Nutch, Packt Publishing, Birmingham, UK, 2013.

[32] P. Willett, The porter stemming algorithm: then and now, Program 40 (3) (2006) 219–223.

[33] H.V. Karambelkar, Scaling Apache Solr, Packt Publishing Ltd, 2014.

[34] Most popular web application frameworks, (http://www.hurricanesoftwares.com/most-popular-web-application-frameworks/).

[35] T. Thai, H. Lam, .NET Framework Essentials, O'Reilly Media Inc., Sebastopol, CA, USA, 2003.

[36] J. Blanchette, M. Summerfield, C++ GUI Programming with Qt 4, Prentice Hall Professional, Westford, MA, USA, 2006.

[37] D.M. Geary, Graphic Java 2: Mastering the JFC, Prentice Hall, Palo Alto, CA, USA, 1999.

[38] 53 percent of organic search clicks go to first link, (http://searchenginewatch.com/sew/study/2215868/53-of-organic-search-clicks-go-to-first-link-study).

[39] S. Easterbrook, J. Singer, M.-A. Storey, D. Damian, Selecting empirical methods for software engineering research, in: Guide to Advanced Empirical Software Engineering, Springer, London, UK, 2007, pp. 285–311.

[40] M.P. Robillard, W. Maalej, R.J. Walker, T. Zimmermann, Recommendation Systems in Software Engineering, Springer, Berlin Heidelberg, Germany, 2014.

[41] M.-A. Storey, Theories, tools and research methods in program comprehension: past, present and future, Softw. Qual. J. 14 (3) (2006) 187–208.

[42] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, C. Fu, Portfolio: finding relevant functions and their usage, in: Proceedings of the 33rd International Conference on Software Engineering, IEEE, 2011, pp. 111–120.

[43] H. Zhong, T. Xie, L. Zhang, J. Pei, H. Mei, MAPO: mining and recommending API usage patterns, in: Proceedings of the 23rd European Conference on Object-Oriented Programming, in: vol. 5653 of Lecture Notes in Computer Science, Springer, 2009, pp. 318–343.

[44] F. Thung, S. Wang, D. Lo, J. Lawall, Automatic recommendation of API methods from feature requests, in: Proceedings of the 28th International Conference on Automated Software Engineering, IEEE, 2013, pp. 290–300.

[45] R. Holmes, R.J. Walker, G.C. Murphy, Strathcona example recommendation tool, ACM SIGSOFT Softw. Eng. Notes 30 (5) (2005) 237–240.

[46] N. Sahavechaphan, K. Claypool, XSnippet: mining for sample code, ACM Sigplan Not. 41 (10) (2006) 413–430.

[47] S. Thummalapenta, T. Xie, PARSEWeb: a programmer assistant for reusing open source code on the web, in: Proceedings of the 22nd International Conference on Automated Software Engineering, IEEE/ACM, 2007, pp. 204–213.

[48] S. Thummalapenta, T. Xie, SpotWeb: detecting framework hotspots and coldspots via mining open source code on the web, in: Proceedings of the 23rd International Conference on Automated Software Engineering, IEEE/ACM, 2008, pp. 327–336.

[49] Y. Ye, G. Fischer, B. Reeves, Integrating active information delivery and reuse repository systems, ACM SIGSOFT Softw. Eng. Notes 25 (6) (2000) 60–68.

[50] Y. Ye, G. Fischer, Supporting reuse by delivering task-relevant and personalized information, in: Proceedings of the 24th International Conference on Software Engineering, ACM, 2002, pp. 513–523.

[51] R. Hoffmann, J. Fogarty, D.S. Weld, Assieme: finding and leveraging implicit references in a web search interface for programmers, in: Proceedings of the 20th Annual Symposium on User Interface Software and Technology, ACM, 2007, pp. 13–22.

[52] B. Dagenais, H. Ossher, Automatically locating framework extension examples, in: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2008, pp. 203–213.

[53] S. Chatterjee, S. Juvekar, K. Sen, SNIFF: a search engine for java using free-form queries, in: Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering, Springer, 2009, pp. 385–400.

[54] S.P. Reiss, Semantics-based code search, in: Proceedings of the 31st International Conference on Software Engineering, IEEE, 2009, pp. 243–253.

[55] L. Moreno, G. Bavota, M.D. Penta, R. Oliveto, A. Marcus, How can i use this method? in: Proceedings of the 37th International Conference on Software Engineering, IEEE, 2015, pp. 880–890.

[56] J. Stylos, B. Myers, et al., Mica: a web-search tool for finding API components and examples, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE, 2006, pp. 195–202.

[57] K.T. Stolee, S. Elbaum, M.B. Dwyer, Code search with input/output queries: generalizing, ranking, and assessment, J. of Sys. and Soft. 116 (2016) 35–48.

[58] B. Dagenais, H. Ossher, Aiding evolution with concern-oriented guides, in: Proceedings of the 3rd Workshop on Linking Aspect Technology and Evolution, ACM, 2007.