Recommended Component by Citation: A Semi-supervised Approach for Determination

Sibo Cai, Yanzhen Zou, Lijie Wang, Bing Xie, Weizhong Shao
Software Institute, School of EECS, Peking University, Beijing, P.R. China
Key Laboratory of High Confidence Software Technologies,
Ministry of Education, Beijing, P.R. China
{caisb06, zouyz, wanglj07, xiebing}@sei.pku.edu.cn, wzshao@pku.edu.cn

Abstract—Reusing existing components can help developers improve the development productivity as well as reduce the cost. Reuse repositories in this scenario act as a fundamental facility for acquiring needed components. While retrieving components in reuse repositories, developers often face the problem of choosing components from candidates that provide similar functionalities. To address the problem, this paper proposes a semi-supervised method to recommend developers components in reuse repositories. Different from existing rating based recommendation approaches that often suffer from the lack of user ratings, our approach calculates the recommendation probabilities of components based on their citations on the Internet. The citations are acquired through the websites (called host in this paper) that are associated with the components. Using a random walk algorithm, the associations between components and hosts are explored with recommendable components identified. We implemented our approach in a prototyping system based on which we conducted an experimental study to evaluate our approach. The experimental results demonstrate that our approach can accurately recommend components and thus has the potential to assist developers in reuse.

Keywords—software reuse; component recommendation; reuse repository

I. INTRODUCTION

It is widely believed that reusing existing components can help developers create applications with less effect and improved quality [1]. To find proper components, developers need to retrieve reuse repositories according to their reuse plan [2]. Generally, the component retrieval process can be divided into two steps: Firstly, developers search reuse repositories to find component candidates that provide needed functionalities. Secondly, developers select components from candidates and integrate them together to build the target system.

Selecting components from candidates is very crucial for system integration. Before the adaption and integration of the components, developers often have to prototype a testing environment to validate the selected components to make sure that the selected components could satisfy the detailed requirements [3]. A casual selection of components may put developers into the risk of wasting a lot of time and effort. The situation becomes even worse if the components are not cost-free.

Currently, most widely used mechanism to provide hints for developers to make informed component selection decision in real world reuse repositories, such as SourceForge [5] and ComponentSource [6], is the user rating/review system. In such systems, developers can share their experience of certain component by simply giving a rating or a paragraph of remarks. Then all the ratings are aggregated for providing decision information to assist developers with components selection. User rating/review systems can provide information to assist developers in making selection decision; however, this kind of mechanisms is always blamed for their limited ratings due to the user motivation problem [7]. In most cases, developers have to spend extra effort collecting related information or try the retrieved candidates one by one to decide which component to use. The retrieval process becomes inefficient.

To resolve the problem, we propose a novel component recommendation method based on the citations of components appearing on the Internet. The citations of components on the Internet can be components for download (such as the components in Download.com [21]), components at runtime (like Java Applet [9]) or component-centric description or discussion (such as the appearance in Ohloh [22]). No matter what cases, components appearing on the Internet would relate to certain hosts. In general, the more times a component appears on the Internet, the more probable it should be recommended to developers [8]. Furthermore, if a host is known to involve amounts of recommended components, components cited by this host are more likely to be recommendable ones. Therefore, a component’s citation on the Internet can be utilized for recommendation instead of user ratings.

In this paper, we propose a semi-supervised approach for component recommendation in reuse repositories based on their citations on the Internet and implement the approach in a prototyping system. In the approach, we obtain the hosts that involve the components from the Internet and build the associations between the components and the hosts. Through exploring the associations with a random walk algorithm, we work out the recommendation probability for each component. We evaluate our approach through an experimental study based on our implemented system and real world data. The results show that our approach can accurately identify recommendable components.

1Corresponding Author
The rest of this paper is organized as follows: Section 2 describes our approach for recommending components and an implemented prototyping system. Section 3 presents our experimental study on the proposed approach with the results analyzed. Section 4 presents some discussions about the approach and the future work. Section 5 describes the related works. Then in the last section, we conclude this paper.

II. OUR APPROACH

In this section, we will introduce our approach for recommending components as well as an implementation of the approach based on which we conducted the experimental study. The approach consists of following steps.

- Obtaining hosts associated with components. This step aims to find hosts related to the components from the Internet and build associations between them.
- Setting up the association weight. The relevance degree of components with their hosts is considered in our approach. For each association, a weight denoting the relevance degree is assigned for further computation.
- Computing component recommendation probability. This step computes the recommendation probability for each component through exploring the associations using a random walk algorithm and finally identifies the recommendable components.

Details of these steps are described below.

A. Obtaining Hosts associated with Components

To obtain the hosts related to the components from the Internet, we use the web search engine Google\(^1\) to accomplish the hosts crawling task. We use Google to search the components with component name as the query and extract URLs from the returned result list. Since the most relevant results are usually distributed at the top position, we only keep the top 30 URLs for each component (returned list less than 30 URLs will all be kept). Then the hosts related to the component are identified (we use the domain part of a URL to represent a host. URLs with the same domain part will be merged).

However, the returned results by Google do not always refer to the exact component if we directly use the component name as the query. In order to obtain the hosts and build the associations more accurately, some strategies are adopted to refine the associations between the components and the hosts. Firstly, Google in general will split the input keywords into word segments, search each segment separately and at last merge the results of each word segment to produce the final results. Such manner, however, is intended to fetch web pages that contains word segments of the query but in fact are irrelevant to the components in our scenario. To solve the problem, we validate the returned results by using full match of the component name in both the title and the snippet. This can be fulfilled by adding quotation mark to the component name when searching components. Secondly, the Google web search engine aims to obtain as diverse results as possible, but in our scenario, the returned results by Google should be restricted in the domain of “Software” or “Software Development”. For this problem, we append the keyword “Software” to the query to refine the returned results.

Formally, the associations between components and their hosts can be described as a bipartite graph, \(G = \langle \text{COMP, HOST, EDGE} \rangle\). The component set COMP and the host set HOST constitute the partitions of the graph. Every association is described as an edge \((\text{comp}, \text{host}) \in \text{EDGE}\) where \(\text{comp} \in \text{COMP}\) and \(\text{host} \in \text{HOST}\). Fig. 2(a) gives an example of a bipartite graph.

B. Setting up the Association Weight

In our approach, we consider the degree of components associated with their hosts and a weight is assigned to each association. The association weight between a component and a host denotes the relevance degree of the component to the host. In general, the more times a component appears on one host, the more degree it is relevant to the host.

The association weight of a component related to a host is firstly set as the number of URLs related to the component that are merged into the host. Then, the weights of all associations are normalized using (1) where \(N_{\text{comp-host}}\) is the number of URLs related to the component \(\text{comp}\) and merged into \(\text{host}\). Since the association is non-directional, the weight of a component related to a host equals to the one in reverse order.

\[
w_{\text{comp-host}} = w_{\text{host-comp}} = \frac{N_{\text{comp-host}}}{\sum_{c,h:(c,h)\in \text{EDGE}} N_{c-h}}
\]

C. Computing Component Recommendation Probability

The idea behind our component recommendation approach is that the more hosts by which a component is cited, the more probable a component is to be recommended. More importantly, if a host is known to involve amounts of recommended components, components appearing on the host are more likely to be recommendable ones. To implement the idea, we assign each component a recommendation probability (called “component recommendation probability” in this paper), which means the degree of the component to be recommended. We also set a weight (called “host recommendation probability” in this paper) to every host. The host recommendation probability indicates the degree of a host involving recommended components.

Let \(P(\text{host})\) denotes the host recommendation probability of \(\text{host}\) and \(P(\text{comp})\) denotes the component recommendation probability of \(\text{comp}\). Clearly, a host that involves more components with high component recommendation probability will gain higher host recommendation probability. We use (2) to calculate the host recommendation probability of \(\text{host}\) where \(C(\text{host})\) denotes the components cited by \(\text{host}\).

\[
P(\text{host}) = \sum_{\text{comp}: \text{comp} \in C(\text{host})} w_{\text{host-comp}} P(\text{comp})
\]
On the other hand, a component that is involved by more hosts with high host recommendation probability can also gain higher component recommendation probability. Hence, we use (3) to update the component recommendation probability of a component where \( R(\text{comp}) \) denotes the hosts that involve \( \text{comp} \).

\[
P(\text{comp}) = \sum_{\text{host} \in R(\text{comp})} w_{\text{comp-host}} P(\text{host})
\]

The starting point of the computation is a set of components that are prejudged to be recommended ones and assigned high recommendation probability. We call this group of components “seeds”. To implement the computation, we propose a propagation algorithm by adopting a random walk algorithm with absorbing states [14]. The algorithm is presented in Fig. 1.

**Input:** the seed set \( S \), COMP-HOST bipartite graph \( G \), the vanishing threshold \( \beta \), the transition probability \( \alpha \) to \( \omega \)  
**Output:** \( P(\text{comp}) \), for every component except for the seeds  
1: for each \( \text{comp} \) in \( S \) do \( P(\text{comp})=1 \)  
2: repeat  
3: for each \( \text{host} \) in HOST do  
4: \( P(\text{host}) = (1-\alpha) \sum_{\text{comp} \in R(\text{host})} w_{\text{comp-host}} P(\text{comp}) \)  
5: if \( P(\text{host}) < \beta \) then \( P(\text{host}) = 0 \)  
6: end for  
7: for each \( \text{comp} \) in COMP \( S \) do  
8: \( P(\text{comp}) = (1-\alpha) \sum_{\text{host} \in R(\text{comp})} w_{\text{comp-host}} P(\text{host}) \)  
9: if \( P(\text{comp}) < \beta \) then \( P(\text{comp}) = 0 \)  
10: end for  
11: until convergence

**Figure 1.** The propagation algorithm

The input of the algorithm contains the seed set \( S \), COMP-HOST bipartite graph \( G \) and two parameters \( \alpha \) and \( \beta \). The transition probability \( \alpha \) denotes that both the components and the hosts have the probability to transfer to the absorbing state \( \omega \) [14]. This means that if a component or a host does not keep acquiring the recommendation probability, its probability will gradually decrease to 0 and thus be absorbed by \( \omega \). The vanishing threshold \( \beta \) is used as the threshold to distinguish the recommendable components. The output of the algorithm is the component recommendation probability \( P(\text{comp}) \) for each component except the seeds.

Initially, the component recommendation probabilities of the seeds are set to 1 (Line 1). Then the algorithm (Line 3 to Line 10) iteratively calculates the probability for both the hosts and the components. In each iteration the host recommendation probability for each host is calculated using (2) and weakened by \((1-\alpha)\) (Line 4). If the calculated probability is less than \( \beta \), the probability is set to 0. The calculation of the component recommendation probability is similar to the one of host recommendation probability. Note that besides the components, we also use \( \beta \) as a threshold to discard the probabilities of hosts in Line 5. This is mainly for efficiency purpose [14]. The iteration will continue until the recommendation probabilities of both the components and the hosts are convergent. Components with the recommendation probability greater than 0 then will be regarded as recommended ones.

To explain the propagation algorithm in a more concrete way, an illustrated example is given in Fig. 2. In the example, there are 5 components (indicated as squares) and 4 hosts (indicated as circles) that have associations with the components as shown in (a). Components C2 and C3 are seeds. Suppose that all the associations have the same weight. Initially, the component recommendation probabilities of C2 and C3 are set to 1. In the first iteration, hosts that have associations with C2 and C3 will gain host recommendation probability. Thus, the host recommendation probabilities of H1, H3 and H4 (indicated as solid circles) are updated as shown in (b). Note that the host recommendation probability of H3 will be larger than those of H1 and H4 since H3 has associations with both C2 and C3. The hosts will also backwards affect the recommendation probabilities of the components except the seeds. Therefore, the component recommendation probabilities of C4 and C5 (indicated as solid squares) will be updated as shown in (c). Similarly, the host recommendation probabilities of all the 4 hosts (shown in (d)) and backwards the component recommendation probabilities of C1, C4 and C5 will be updated (shown in (e)) in the following iteration. The iteration continues until the recommendation probabilities of both the components and the hosts converge.

**Figure 2.** An illustrated example of the propagation algorithm

\( \text{D. Implementation with a Prototyping Reuse Repository} \)

We implement our approach with a prototyping reuse repository that simply provides free-text based component retrieval [10]. Note that although we use free-text approach to acquire relevant components in the prototyping system, our method for recommending components is not limited by the component retrieval mechanisms. The architecture of the reuse repository with our recommendation approach is presented in Fig. 3. Components are stored in the \textit{retriever}. The \textit{retriever} accepts queries of developers and retrieves relevant
component candidates from the reuse repository for further recommendation.

The implementation of our approach is denoted in the dash line rectangle in Fig. 3. The crawler obtains the hosts that have association with the components in the reuse repository from the Internet and stores the associations in the association database. Based on the work done by the crawler, the analyzer computes the recommendation probability for each component utilizing the associations and produces the recommendable components based on the retrieved candidates to developers.

![Diagram of the architecture of the reuse repository with our recommendation approach](image)

**Figure 3.** The architecture of the reuse repository with our recommendation approach

### III. EXPERIMENTAL STUDY

#### A. Experimental Organization

To evaluate our approach, we applied it to the data collected from a real world reuse repository, i.e. SourceForge, which not only provides software systems, but also many reusable libraries that fuel the further development of new applications. Particularly, we selected the category “Software Development” as our evaluation base. Category “Software Development” totally contains 35,602 software projects at the time we carried out the evaluation. We acquired the project information including project name, description, user positive rating and negative rating etc. by implementing a web page crawler. The 35,602 software projects constituted the reuse repository in our approach. In our experiment, each software project was viewed as a component.

Based on the built reuse repository, we used Google to search for the hosts that involve the components in the reuse repository and extracted associations between the components and the hosts. In this study, we removed the SourceForge from the host set to eliminate the impact of SourceForge since SourceForge acts as the evaluation base of our experimental study. Associations including SourceForge were also excluded. Thus, we totally fetched 251,873 hosts as well as 937,016 associations between the component set and host set.

We sorted the 35,602 projects according to the user ratings provided in our crawled pages from SourceForge. After a review of the sorted list, 100 projects were selected as seeds (less than 0.3% of the component set). The 100 projects were either famous software projects, or with high user positive rating that are supposed to recommend to developers.

In the execution of the propagation algorithm, we also need to set the parameters, i.e. $\alpha$ and $\beta$. In our experiment, $\alpha$ was set to an empirically small value 0.01 [14] while the selection of $\beta$ somewhat depends on the application scenarios. To solve the selection of $\beta$, we used the seed set to tune it. We divided the seed set into 2 parts. Four fifth of the seed set were still used as seeds in the propagation algorithm while the remaining one fifth were regarded as the test set to tune $\beta$. The tuning algorithm is shown in Fig. 4. After conducting the tuning algorithm, we finally set $\beta$ to 5E-5.

![Tuning Algorithm for $\beta$](image)

**Figure 4.** The tuning algorithm for $\beta$

#### B. Experimental Results

We evaluated the effectiveness of our approach by using developer queries. In order to decide which queries to use, we interviewed 7 graduate students in Peking University who have more than 2 years of software development experience. Finally, 11 queries were identified. We then submitted the 11 queries to our prototyping system to retrieve relevant components and more importantly the recommended ones by our approach. To validate the recommended components, we adopted the user ratings provided by SourceForge. Components were regarded as recommended ones if their number of positive ratings in SourceForge is larger than the one of negative ratings.

<table>
<thead>
<tr>
<th>Query</th>
<th>Recommended</th>
<th>Retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Parser</td>
<td>3(5)</td>
<td>14</td>
</tr>
<tr>
<td>Data Encryption</td>
<td>1(1)</td>
<td>7</td>
</tr>
<tr>
<td>Logging</td>
<td>9(11)</td>
<td>12</td>
</tr>
<tr>
<td>Math</td>
<td>5(5)</td>
<td>12</td>
</tr>
<tr>
<td>Statistics</td>
<td>5(5)</td>
<td>7</td>
</tr>
<tr>
<td>Data Compression</td>
<td>1(1)</td>
<td>10</td>
</tr>
<tr>
<td>Email</td>
<td>1(1)</td>
<td>6</td>
</tr>
<tr>
<td>File Upload</td>
<td>2(4)</td>
<td>14</td>
</tr>
<tr>
<td>Configuration File</td>
<td>3(2)</td>
<td>11</td>
</tr>
<tr>
<td>Network Utility</td>
<td>1(1)</td>
<td>11</td>
</tr>
<tr>
<td>IO Utility</td>
<td>2(2)</td>
<td>14</td>
</tr>
</tbody>
</table>

To evaluate our approach, we identified the recommended components by SourceForge from the retrieved ones for each query and then found out whether our approach could produce similar recommended list. The results are presented in TABLE I. The queries are listed in the “Query” column. Column “Retrieved” indicates the number of retrieved components according to the query while the “Recommended” column
represents the number of recommended components by our approach contained in the recommended list of SourceForge. The number of components recommended by SourceForge is also indicated (in the bracket). Take query “XML Parser” as an example, the number of retrieved components related to the query is 14 and in the 5 recommended components of SourceForge, our approach suggests 3 of the 5.

Through the comparison with recommended components by SourceForge that are based on user ratings, we preliminarily drew to the conclusion that our approach possesses the ability to identify the recommended components suggested by SourceForge. More queries should be performed to further validate our approach; however, the selected queries to some extent cover amply portion of fundamental software development needs based on our interview with software developers. In order to show the effectiveness of our approach in a better fashion, seeds (if retrieved) were excluded in the retrieval results.

We also evaluated the precision of the recommended components by our approach. Precision here means the ratio of actual recommended components compared to the ones suggested by our approach. We compared the number of recommended components by our approach to the ones suggested by SourceForge. The 11 queries were still used. TABLE II shows the comparison results. In the column “Recommended”, the number of recommended components of our approach is indicated in the bracket. In the recommended list of our approach, the number of components that are also suggested by SourceForge is indicated outside the bracket. In half cases, our approach performs well and produces recommended components similar to the ones by SourceForge, such as “Logging”, “Statistics”. While in the other half, our approach recommended more components than SourceForge does, especially in the case “Network Utility”.

TABLE II. OUR RECOMMENDED COMPONENTS COMPARED TO THE ONES BY SOURCEFORGE

<table>
<thead>
<tr>
<th>Query</th>
<th>Recommended</th>
<th>Retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Parser</td>
<td>3(7)</td>
<td>14</td>
</tr>
<tr>
<td>Data Encryption</td>
<td>1(3)</td>
<td>7</td>
</tr>
<tr>
<td>Logging</td>
<td>9(9)</td>
<td>12</td>
</tr>
<tr>
<td>Math</td>
<td>5(8)</td>
<td>12</td>
</tr>
<tr>
<td>Statistics</td>
<td>5(6)</td>
<td>7</td>
</tr>
<tr>
<td>Data Compression</td>
<td>1(3)</td>
<td>10</td>
</tr>
<tr>
<td>Email</td>
<td>1(5)</td>
<td>6</td>
</tr>
<tr>
<td>File Upload</td>
<td>2(7)</td>
<td>14</td>
</tr>
<tr>
<td>Configuration File</td>
<td>3(7)</td>
<td>11</td>
</tr>
<tr>
<td>Network Utility</td>
<td>1(8)</td>
<td>11</td>
</tr>
<tr>
<td>IO Utility</td>
<td>2(6)</td>
<td>14</td>
</tr>
</tbody>
</table>

To explore the reason of this phenomenon, we investigated the recommended components by our approach while not suggested by SourceForge. We found that almost all these components received 0 positive rating and 0 negative rating in SourceForge. According to the criteria of recommended components, such components will be judged as not recommended by SourceForge. “GSA Simple XML Parser” by hand, we finally judged that this component should also be recommended.

IV. DISCUSSION AND FUTURE WORK

A. Issues about the Association Refinement

In our approach, we refine the association between components and hosts using the strategies described in section 2, but there are still some problems in building the associations. The most extrusive case is that the name of a component is too general. For example, an xml parser named “xml parser”. To search such keywords in Google, the returned results are often irrelevant to the component. Such cases will reduce the effectiveness of our approach and provide problematic recommended list to the developers. Nevertheless, we find that such examples only occupy a very small fraction of the components in real world reuse repositories since people who develop components are mostly intended to pick up a more meaningful name for their components while further consideration should be taken to deal with such cases.

B. How to Obtain the Seed Set

In our approach, one of the inputs to calculate the recommendation probabilities of the components is the seed set. The effectiveness of our approach will be greatly reduced if the seed set is hard to obtain. However, the seed set seems not so difficult to identify in real world reuse repositories. Firstly, just like our experimental study, components that have already received high user positive ratings can be considered. Secondly, famous software components are another option. Thirdly, components that developed by famous companies or organizations can also be taken into consideration.

Another issue about the seed set is how to construct a better seed set. The selection of the seeds may influence the performance of our approach since it is the starting point. Selecting seeds as divergent as possible may be one of the possible strategies that can be used to enhance the performance of our approach. For instance, seeds can be selected considering different application domains. Further study will be carried out in our future work.

V. RELATED WORKS

Helping developers obtain appropriate components is crucial for successful software reuse. In the literature, many research works have been studied to facilitate the component retrieval, such as 1) free text approaches [10]; 2) facet based approaches [11]; 3) signature based approaches [12] and 4) behavior based approaches [13]. Most of these research works concentrate on obtaining relevant components while pay little attention to providing information for developers choosing components from candidates that provide similar functionalities. As the size of reuse repositories becomes large, helping developers make informed decision among functionally similar components can accelerate the process of selection and this is what our approach is trying to accomplish.

To recommend software components among candidates, Inoue et al. proposed the component rank model based on which they developed SPARS-J, a java class retrieval system

---

2 http://sourceforge.net/projects/gsa-simple-xml/
The component rank model analyzes the usage relation of components so as to identify recommendable components that are used more frequently. The work is based on the premise that the component source code is acquirable but in many cases (e.g. COTS) this is unrealistic. Ichii et al. proposed a software component recommendation approach based on collaborative filtering (CF) utilizing user browsing history [15]. However, CF inborn suffers from the “Cold Start” problem. In real world reuse repositories, such as SourceForge and ComponentSource, user rating/review systems are usually used to recommend software components while these systems are often blamed for their shortage of ratings/reviews [7]. In our approach, we make use of the associations between components and hosts obtained from the Internet to avoid the above problems as well as provide hints to help developers make selection decision.

There are also several pieces of work concerning recommendation systems in the research area of software engineering. CodeBroker [16] provides recommendation of APIs for developers that may implement the needed functionalities the moment developers write down comments or the signature of methods. Strathcona [17] developed by Holmes et al. recommends example code to developers by monitoring the code under development. The recommendation is made according to six structure-based heuristics. Li et al. proposed an approach to recommend typical usage example of APIs by adopting code clustering [18]. Kim et al. presented an approach that searches code bases on the Internet and extract code examples to insert into API documents to recommend developers suitable examples while using certain APIs [19]. ParseWeb [20] is another useful tool to suggest call sequences for developers when developers are using certain APIs. Different from these approaches that recommend implemented APIs or example code, our approach aims to help developers make informed choice facing components that provide similar functionalities.

VI. CONCLUSION

In this paper, we proposed a semi-supervised approach to produce recommended components to the developers to assist their selection of components in reuse repositories. The approach utilizes the associations between the components and the involved hosts. Through a selected group of components that are supposed to be recommended and a propagation algorithm, the recommendation probability for each component is calculated. We also implemented a prototyping system and conducted an experimental study on our approach using real world data. The results show that our approach can accurately recommend components to the developers comparing to the data from SourceForge.

ACKNOWLEDGMENT

We would like to thank Jing Jin for the experimental data collection. This research was sponsored by the National Natural Science Foundation of China under Grant No. 60821003, the National Basic Research Program of China (973) under Grant No. 2009CB320703 and the National High-Tech Research and Development Plan of China under Grant No. 2007AA010301-01.

REFERENCES


