Maintaining Third-Party Libraries through Domain-Specific Category Recommendations

Daiki Katsuragawa, Akinori Ihara, Raula Gaikovina Kula, Kenichi Matsumoto
Nara Institute of Science and Technology, Japan
{katsuragawa.daiki.ka7, akinori-i, raula-k, matumoto}@is.naist.jp

ABSTRACT
Proper maintenance of third-party libraries contributes toward sustaining a healthy project, mitigating the risk it becoming outdated and obsolete. In this paper, we propose domain-specific categories (i.e., grouping of libraries that perform similar functionality) in library recommendations that aids in library maintenance. Our empirical study covers 2,511 GitHub projects and 150 domain-specific categories of Java libraries. Our results show that a system uses up to six different categories in their dependencies. Furthermore, recommending domain-specific categories is practical (i.e., with an accuracy between 66% to 81% for multiple categories) and its suggestion of libraries within that domain is comparable to existing techniques.

ACM Reference Format:

1 INTRODUCTION
Software libraries play an important role in the health of a software project, especially in terms of its success, longevity, growth, resilience, survival, diversity, and sustainability. Two cited reasons for modern Open Source project failures are the risk of becoming obsolete (i.e., no longer useful) and continued usage of outdated technologies (i.e., to outdated, deprecated or suboptimal technologies, including programming languages, APIs, libraries, frameworks, and so on) [7]. In fact, recent studies show that outdated libraries are commonplace, with the potential to hinder project growth while risking exposure to vulnerabilities [1, 3, 8, 15, 16, 19]. Often, libraries that we depend on for a larger software system become dormant; its development ceases. As operating systems, deployment frameworks and security infrastructure evolve, there is a likelihood that a software will break because of a dormant library dependency.

We conjecture that searching for useful libraries stretches out the life-span (i.e., functionality, appeal and usability) of an application. For instance, a web-based application’s life-span would be rejuvenated with the help of specialized libraries like GWT\(^1\), Spring\(^2\), Hibernate\(^3\) to expand its current set of features.

To cope with the search and maintenance of libraries, existing research leverages software library recommendation systems. For example, Thung et al. [26] proposed a technique that automatically identifies new candidate libraries to unaware developers. We speculate a limitation—that existing techniques ignore that candidate libraries may belong to the same domain. To address this limitation, we introduce domain-specific categories (DSC), as a classification of software libraries based on their specific functional properties or domains. We conjecture that popular libraries belonging to the same domain may be of interest to developers. Libraries may serve a specific functionality such as a logging framework (i.e., log4j), HTML analysis (i.e., jsoup) and SSH (Secure Shell) and encryption (i.e., bouncycastle). For instance, library search services\(^4\) use these categories to search and discover new libraries.

In this paper, we investigate how DSC aids library recommendation. Using association rule mining, we conducted an empirical study that covers 2,511 GitHub projects to investigate the diverse usage of DSCs. Results of the study show that projects depend on multiple libraries that belong to various categories, with systems depending on up to 6 different types of DSC in their library dependencies. Our approach uses association rule mining to show how our technique is practical, with an accuracy of 66% to 81% for multiple categories. However, we show in a comparative study that our prototype DSCRec is comparable to existing techniques (i.e., LibRec[26]).

Our main contributions are two-fold. The first contribution is an investigation into DSC and how they contribute to library recommendation. The second contribution is evidence that although our technique is practical, its effectiveness is not as straightforward as selecting the most popular library within that domain.

2 MOTIVATION & RESEARCH OVERVIEW
First, we describe our motivation and the problem definition. We then introduce our study and our research questions.

2.1 Problem Definition & Illustrative Example
We adopt the exact problem definition of library recommendation from Thung et al., which is that it should satisfy two conditions [26]:

\(^1\)GWT: http://www.gwtproject.org
\(^2\)Spring: http://spring.io/
\(^3\)Hibernate: http://hibernate.org/
\(^4\)Maven Repository at https://mvnrepository.com/
(1) it should not contain an existing library.
(2) it should be useful.

To satisfy the second condition, Thung et al. proposed a hybrid technique of mining the usage patterns of libraries to reveal more useful libraries. As well as library popularity and usage patterns, domain-specific categories has potential to also reveal useful libraries. For instance, current techniques would reveal the specific library such as JUnit (i.e., which is a testing library). In our approach, we present any libraries within the testing framework category. As shown at the Maven Repository website, there are up to 42 different Maven testing frameworks available to developers. Under this higher level of abstraction, other testing libraries such as testNG can also be presented as viable candidates to the developer. As a result, not only recommending DSC, our approach suggests candidate libraries within the DSC.

To achieve this, our approach follow two steps. It first presents a domain-specific category. After this step is completed, our technique suggests a candidate library belonging to that specific domain.

### 2.2 Research Questions

In this study, we investigate the practical implications of using DSC for library recommendations. As shown in Figure 1, we formulate three research questions that guide our study.

- **RQ1**: How diverse are the specific domains of libraries adopted by a software system? The motivation of the first research question is to understand whether or not projects consist of a wide range of DSC in their dependencies.

- **RQ2**: How accurate is the recommendation of domain-specific categories? The motivation of the second research question is to investigate domain-specific categories practical usage in recommendation models (i.e., category recommendation).

- **RQ3**: How do domain-specific categories impact library recommendation? The motivation of the final research question is to investigate how domain-specific categories are effective for library recommendations.

### 3 DATA EXTRACTION

To answer our research questions, we created a dataset that captures the DSC of library dependencies. Our dataset comprises of (1) a set of projects that use Maven Libraries (i.e., Kula et al. [16]) and (2) a labeled set of domain-specific categories of Maven Libraries (i.e., Maven Repository website). We performed two major activities of (1) extraction of systems and libraries and (2) mapping the categories to the libraries:

**Extraction of Target Systems and their Libraries.** Similar to related work [18, 21, 26], library usage is extracted from the listed dependencies in the pom.xml. We use the dataset provided by Kula et al. [16]. We use the following dependency properties: <groupId>

![](https://mvnrepository.com/open-source/testing-frameworks)

indicating the developer name, <artifactId> indicating the library unique id, and <version> indicating the version of the library. It is important to note that a project may contain more than one system (i.e., a project may contain several pom.xml files).

To ensure a quality dataset, we applied filtering to remove noisy systems from our dataset (i.e., such as single dependencies). Additionally, we targeted more mature and complex projects with more complex library dependencies. Similar to Thung et al., we targeted systems that use at least ten or more libraries.

**Mapping DSC to Libraries.** In this step, we mapped the targeted system libraries to domain-specific categories. As mentioned in the prior steps, we extracted a labeled set of domain-specific categories with their libraries from the Maven Repository website. As shown in Figure 1, then mapped the libraries to each category. Note that libraries that not mapped to any domain-specific category are classified as “others”. Consistent with our goal to collect mature and complex projects, we filtered out target systems with a single domain-specific category.

From an original dataset of 8,142 systems, after filtering and mapping of categories, we were left with the final dataset of 7,185 systems and 38,848 libraries. We mapped 150 domain-specific categories to support finding appropriate library (i.e., see Table 1). As shown in the Table, the most popular (i.e., calculated by the frequency) domain-specific categories are Testing Frameworks, Logging Frameworks, and Language Runtime. To validate the coverage of the dataset, as shown in Table 2, we show that up to 94% of the top 100 popular libraries is classified into any domain-specific categories.

<table>
<thead>
<tr>
<th>Rank</th>
<th>DSC</th>
<th>FR</th>
<th># systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Testing Frameworks</td>
<td>52%</td>
<td>4,198</td>
</tr>
<tr>
<td>2</td>
<td>Logging Frameworks</td>
<td>48%</td>
<td>3,940</td>
</tr>
<tr>
<td>3</td>
<td>Java Specifications</td>
<td>42%</td>
<td>3,437</td>
</tr>
<tr>
<td>4</td>
<td>Core Utilities</td>
<td>39%</td>
<td>3,181</td>
</tr>
<tr>
<td>5</td>
<td>Logging Bridges</td>
<td>29%</td>
<td>2,348</td>
</tr>
<tr>
<td>6</td>
<td>Dependency Injection</td>
<td>27%</td>
<td>2,162</td>
</tr>
<tr>
<td>7</td>
<td>JSON Libraries</td>
<td>26%</td>
<td>2,154</td>
</tr>
<tr>
<td>8</td>
<td>Mocking</td>
<td>25%</td>
<td>2,048</td>
</tr>
<tr>
<td>9</td>
<td>I/O Utilities</td>
<td>21%</td>
<td>1,729</td>
</tr>
<tr>
<td>10</td>
<td>XML Processing</td>
<td>21%</td>
<td>1,698</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>150</td>
<td>Enterprise Service Bus</td>
<td>0%</td>
<td>5</td>
</tr>
</tbody>
</table>

### 4 EVALUATION

Using the extracted dataset, we proceed to answer our research questions. To answer each research question, we present the approach taken and then present the result, which includes the answer to each question.
Table 2: Percentage of Popular Libraries Classified into DSC Usage Patterns

<table>
<thead>
<tr>
<th>Popular Libraries (Top N)</th>
<th>DSC Usage Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>94%</td>
</tr>
<tr>
<td>200</td>
<td>85%</td>
</tr>
<tr>
<td>300</td>
<td>75%</td>
</tr>
<tr>
<td>400</td>
<td>69%</td>
</tr>
<tr>
<td>500</td>
<td>61%</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>38,884</td>
<td>4%</td>
</tr>
</tbody>
</table>

4.1 RQ1: How diverse are the specific domains of libraries adopted by a software system?

**Approach**. To answer RQ1, we performed an exploratory study of real-world projects and their different categories. We provided two levels of analysis:

- (Step 1) DSC Analysis: In this analysis, we studied (i) the number of libraries used by a system and (ii) the number of different categories per system.
- (Step 2) DSC Usage Pattern Analysis: We used association rule mining to generate some usage pattern rules. Also employed by Thung et al., association rule mining technique is a popular method for the generation of usage rules and patterns [11, 31].

Association rule mining is a method to extract a relationship between two or more items as an association rule from a combination of a large number of items. We use an example of a simple rule showing the relationship that if user has both Logging Frameworks (LF) and Testing Frameworks (TF). In this case, the association rule for both domain-specific categories is represented by pre-condition and pre-condition as follows.

\[ \{ LF \} \Rightarrow \{ TF \} \]  (1)

To evaluate the extracted rules, we used the support, confidence, and lift metrics. We define the support as the proportion of rules which both pre-condition (LF) and post-condition (TF) exist in all systems (i.e., where \( \sigma(LF \cap TF) \) means the number of all systems using both LF and TF). A high support means the rule is a popular combination, while a lower support implies less popularity.

\[ support(LF) \Rightarrow TF = \frac{\sigma(LF \cap TF)}{all \ systems} \]  (2)

The confidence metric is the proportion of rules which both pre-condition (LF) and post-condition (TF) exist in rules with pre-condition (LF). A high confidence means the combination that is likely to be used.

\[ confidence(LF) \Rightarrow TF = \frac{support(LF) \Rightarrow (TF)}{support(LF)} \]  (3)

Finally, the Lift measures the magnification of the data which pre-condition (LF) and post-condition (TF) exist in rules with post-condition (TF). A high lift means strong combination of relationships between the conditions.

\[ lift(LF) \Rightarrow TF = \frac{confidence(LF) \Rightarrow TF}{support(TF)} \]  (4)

We used the Orange [9] library, which uses the apriori algorithm [2] in Python for extraction of the rules. Note that the apriori algorithm used can filters minor rules from the output.
Results. We are able to make the following observations as part of the results to RQ1 (Step 1 and Step 2):

Observation 1 - Systems depend on up to 6 different categories in their library dependencies.

Figure 2 presents the distribution of both the DSC and library usage, confirming that projects are more likely to use multiple categories in their dependencies. Figure 2a shows the distribution of categories per system, showing that systems use up to 6 domain-specific categories. (i.e., median value). Complementary, Figure 2b shows the distribution of libraries used per system, with systems using up to 17 libraries (i.e., median value). Other results are different from related work, with Thung et al. reporting an average or 28 libraries.

Observation 2 - The most common DSC usage pattern is Testing Frameworks DSC and Logging Frameworks DSC.

Confirming the proportions of DSC patterns in Table 1, results from the association rule mining in Table 3 show that Testing Frameworks and Logging Frameworks (i.e., 36% target systems) are the most frequent DSC usage pattern for the target projects. Interestingly, the confidence scores for rule id 1 and 2 in the Table 3 are 0.69 and 0.65, suggesting that both functions are not necessary for usage.

Observation 3 - A system using the Logging Bridges DSC is likely to use a library from the Logging Frameworks DSC, however, this does not necessarily mean that testing and logging frameworks are dependent on each other.

As shown in in Table 3, there are 7 DSC rule pairs (14 rules) which pre-condition and post-condition are interchangeable (i.e., such as rule 1 and rule 2). Using a Fisher exact test, we show that the rules are not coincidental. The Fisher exact test [4, 10] is used to define the interestingness of association rules and has been used in software engineering [28]. As shown in the 7th row, the Testing Frameworks and Logging Frameworks are highly interdependent (i.e., p-value is less than 0.01 for all pairs with a 99% confidence level).

Looking at rule 11 and 12 in Table 3, we find a relatively high number of systems (i.e., 27% systems) uses libraries belonging to both Logging Bridges and Logging Frameworks. Both rules exhibit high support, suggesting that 87% systems with Logging Bridges function are highly likely to use the Logging Frameworks. On the
other hand, 52% systems with Logging Frameworks function use Logging Bridges function, suggesting that although Logging Bridges is often necessary to use Logging Frameworks, Logging Frameworks is not necessary for Logging Bridges. In detail, logging bridges (such as SLF4J, Log4J2, Binding) often control the output of log message. However, this is not needed for some Logging Framework libraries such as SLF4J and Logback.

Based on our results, we now return to answer the first research question:

We find that system depend on multiple libraries that belong to various domain-specific categories. This study shows that up to 8 systems depend on up to 6 different types of domain-specific categories in their library dependencies.

Results from RQ1 provide evidence that using DSC for library recommendation is valid, as systems do contain a diverse set of DSC. Therefore, we proceed to answer RQ2 and RQ3.

4.2 RQ2: How accurate is the recommendation of domain-specific categories?

Approach. To answer RQ2, we built a recommendation model to show that we are able to accurately suggest useful categories. As shown in Figure 1 for the model, our approach trains and evaluates the model is the same as related work [21, 26]. The model also does a comparison against a random guessing model. The training of our model follows these two steps:

- (Step 1) DSC Usage Rules Generation: Similar to RQ1, we mined DSC usage rules from the training dataset. The RuleGenerator accepts the domain-specific category of libraries. The RuleGenerator then generates association rules. To remove noisy rules, we set the minimum support (\(\text{minsup}\)) and minimum confidence (\(\text{minconf}\)) to filter out many minor insignificant rules.

- (Step 2) Ranking of DSC: Algorithm 1 shows the algorithm used to provide a more useful recommendation by ranking the more useful categories. In detail, the algorithm searches for the useful association rules (pre-conditions) included the combination with categories (\(\text{CurDSCat}\)) of the target system (in line 4). When the pre-conditions includes the combination with \(\text{CurDSCat}\), RankGenerator sets DSC with the post-condition in the recommendation list (\(\text{RecList}\)) (in line 9-10). Finally, RankGenerator outputs DSC in \(\text{RecList}\) sorted by confidence score.

To filter out minor rules (i.e., from Step 1), we used the default settings to set \(\text{minsup} = 0.05\) and \(\text{minconf} = 0.4\). We set lower \(\text{minsup} = 0.05\) to avoid missing rules in this experiment. On the other hand, we set a bit higher \(\text{minconf} = 0.4\) to suggest the valued rules.

To evaluate the recommended categories (i.e., from Step 2), we use the well-known Recall Rate@K [20, 22, 26, 29] metric. Let a system be \(S_i\), with at least one of DSC recommended being \(R_i\) and its ground truth \(\{GT_i\}\). Hence, we calculate:

\[
\text{Recall Rate} = \frac{\text{Systems}(S_i|R_i \cap GT_i \neq \varnothing)}{\text{All Systems}}
\]

The Recall Rate for the number of categories (i.e., \(K\) is \(\#\) DSC) is indicated as Recall Rate@K. Importantly, we evaluate the median Recall Rate using a ten-fold cross validation. The control configurations of our recommendations (i.e., \(\text{minsup}\) and \(\text{minconf}\)) indicates usefulness and demand. This study controls \(\text{minsup}\) from 0.05, and \(\text{minconf}\) from 0.20 to 0.65. A higher \(\text{minsup}\) means more demand for this category, while a higher \(\text{minconf}\) suggests a more useful recommendation.

Results. For the results, we first analyze the recall rate for the different DSC recommendation list sizes. We then evaluate the control configurations to understand the usefulness and demand of the recommendation. For this we use the recall Rate@3.

Figure 3 depicts the Recall Rate@K by different recommendation size, showing that accuracy of our model improves as we increase the number of recommended libraries. It shows that the Recall Rate gradually increases as \(K\) increases reaching a peak accuracy of 81%. For instance, Recall Rate@1, Recall Rate@3, Recall Rate@5 and Recall Rate@10 are 41%, 66%, 75% and 81%. Additionally, in the Figure 3, we also show a comparison against a random guessing model, which only gains an Recall Rate of 26% at most (i.e., Recall Rate@10).
Based on our results, we now return to answer second research question:

Our proposed method recommends domain-specific categories with accuracies of Recall Rate@1, Recall Rate@3, Recall Rate@5 and Recall Rate@10 are 41%, 66%, 75% and 81%.

Results from RQ2 show recommending DSC is practical with a reasonable Recall Rate. Therefore, we proceed to RQ3, where we would like to investigate the impact the DSC.

4.3 RQ3: How do domain-specific categories impact library recommendation?

Approach. To answer RQ3, we proposed a library recommendation prototype DSCRec and compare our model to existing techniques. As shown in Figure 1, we adopted the model from RQ2, by adding a step to rank and recommend the most popular library in the recommended DSC list:

- Library ranking by DSC: Our model recommends the most popular library (i.e., calculated by a frequency count of the usage by systems in the dataset). Our key assumption is the popularity of a library within the categories should be the most useful.

We compared our model against two other techniques of (i) LibRec (RULE) proposed by Thung et al. and (ii) random guessing model. We measure performance using the Recall Rate but at the library level (i.e., N is # libraries). Based on Recall Rate@K, we had two parameters of K and N. Thus, the Recall Rate@N at different levels of K.

Based on the RQ2 results, we made the appropriate adjustments to the rank generator (i.e., minsup = 0.05, minconf = 0.4) to allow for library recommendations instead of the category level. We also adjusted the configuration (i.e., minsup = 0.02, minconf = 0.3) to allow for more library recommendations.

Results. For the results, we first analyze the performance of the different models over multiple sets of libraries. Our analysis will present the performance of each model against (i) one library (i.e., N=K) from a category and (ii) two libraries (i.e., N=2K) from multiple categories. Since LibRec does not recommend a category, we only compare the recall rate of recommended library.

Figures 4 shows how LibRec recommends more accurate (higher recall rate) libraries better than our proposed prototype DSCRec. However, as shown by the white portion of the barplot, the recommended DSC portion of the DSC actually performs better than LibRec. This means that the tool is getting the DSC correct, however, fails when selecting the library within the domain. Our naive library recommended method suggests that systems do not necessarily use the more popular libraries within that specific domain.

Based on our results, we now return to answer third research question.

Our proposed prototype DSCRec does not perform better than LibRec. However, the DSC shows better results than LibRec, suggesting that systems do not necessarily use the more popular libraries within the DSC.
5 DISCUSSION

In this section, we discuss the implications of our results, especially taking a closer analysis of the correction of recommendations between DSCRec and LibRec and some qualitative case examples.

Figure 5 shows how correct the models LibRec and DSCRec (DSC and library levels) are against the ground truth, revealing that our method is still comparable to existing techniques. This figure shows that our proposed method could correctly identify 12% of DSC and 6% the libraries not correctly recommended LibRec.

5.1 Practical Examples of Recommendations

We show concrete examples of cases scenarios to understand the strengths and weaknesses of each model output.

Table 4 illustrates the first example where LibRec makes an incorrect recommendation. In this example, our recommended DSC is the correct logging framework. Intuitively, LibRec recommends spring-context, most likely because one of the input libraries is spring-beans, which belongs to the same spring web framework. In this case, the spring-beans does not require the core utilities of the spring-context.

Table 5 illustrates the final example where both LibRec and DSCRec are incorrect, however the DSC was correctly recommended. Under closer investigation, the LibRec recommendation commons-lang belongs to an existing DSC (i.e., core collections). We conjecture that in this case, the filtering of the DSC before the library was not appropriate.

6 THREATS TO VALIDITY

The threats are divided into external, internal and construct validity.

External validity. - refers to the generalization concerns of the study to other library ecosystem such as npm package, RubyGems and the others. This study found that specific results for Java library ecosystem. However, our proposed approach using domain-specific categories contributes a possible solution to recommend appropriate libraries in the other library or package ecosystems. Currently our domain-specific categories only suit Java projects.

Internal validity. - refers to the concerns of definitions of domain-specific categories. First, we rely on the correctness of the domain-specific categories from the Maven Repository website. Based on our experience and by manual evaluation, we are confident that the 150 categories are correct. Furthermore, the study found that the domain-specific category could assist for the library recommendation. However, our target domain-specific categories does not cover all Java systems. As shown in Table 2, the domain-specific categories could cover 94% of the top 100 trend libraries. Therefore we are confident of our domain-specific categories.

Construct validity. - refers to the concerns the construction of the problem definition. Our key assumption is that developers would not like a recommendation that would be a replacement for existing libraries. We understand that there exist cases where a developer would like to know if there are replacements. However, we believe that this is a different kind of recommendation.

7 RELATED WORK

Our related work is separated into three parts: recommendations systems, the use of association rule mining, and work related to libraries. There is been extensive studies that propose different recommendation methods focused on code examples and method-level (i.e. Application Programming Interface (API)). For instance, work by Thummalapenta and Xie [24] proposes ParseWeb, a tool that recommends code examples from a large number of publicly accessible source code repositories. Other work such as Heinemann et al. [12] recommend at the API level. In this work, they propose an approach to recommend API method based on identifier similarity. Other notable API recommendation tools recommend methods based on historical data of code changes [17, 25, 27]. Different to these work, we recommend at the library level, which is a higher abstraction than the API level. Our approach uses the well-known and widely used association rule mining of historical data. For instance, Zimmermann et al. [32] propose an approach to recommend code elements which should change at the same time using association rule mining.

There has been many empirical studies conducted that are related to software libraries, with include library migration and adoption. For instance, Ihara et al. [13] conduct an empirical study to understand the library adaption. Furthermore, work by Teyton et al. [23] propose an approach to visualize library migration graph based on the past library migration. Zerouali et al. [30] and Kabinna et al. [14] also analyzed libraries from the testing category (Testing and Logging Frameworks). Chen [5, 6] propose a tool Similartech that recommend libraries with similar functions by analyzing communications in Stack Overflow when a system migrate from the current library. None of the work use domain-specific categories in their recommendations.

---

Table 4: Example Case 1 where DSCRec makes a correct recommendation and LibRec is incorrect

<table>
<thead>
<tr>
<th>Library</th>
<th>DSCRec</th>
<th>LibRec</th>
</tr>
</thead>
<tbody>
<tr>
<td>commons-lang</td>
<td></td>
<td></td>
</tr>
<tr>
<td>commons-collections</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spring-context</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Example Case 2 where DSCRec makes incorrect library recommendation and correct DSC recommendation

<table>
<thead>
<tr>
<th>Library</th>
<th>DSCRec</th>
<th>LibRec</th>
</tr>
</thead>
<tbody>
<tr>
<td>commons-lang</td>
<td></td>
<td></td>
</tr>
<tr>
<td>commons-collections</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spring-context</td>
<td></td>
<td></td>
</tr>
<tr>
<td>slf4j-api</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

10 The Spring Framework at https://projects.spring.io/spring-framework/
Our study is inspired by the work of Thung et al. [26]. In this work, they propose an approach to recommend libraries by a hybrid approach using combinational association rule mining and collaboration filtering. This approach recommends some libraries that some systems frequently used with a combination. Later, Ouni et al. [21] propose an approach to recommend libraries by a search base algorithm NSGA-II. Related, Mileva et al. [18] propose the tool AKTARI to recommend trend version of a library based on the wisdom of the crowd. We conjecture that as well as popularity, developers would like to identify similar libraries that belong to the same specific domain.

8 CONCLUSION AND FUTURE DIRECTIONS

The maintenance of software libraries plays a key role in keeping a project healthy. To facilitate efficient and effective management of libraries (i.e. update and searching for new candidate libraries), our study investigates the impact of using DSC in library recommendation. Although our proposed library recommendation tool does not perform better than the existing state–of–the–art, there is potential for DSC with library recommendations. For future work, we would to investigate other techniques and combining existing techniques such as collaborative filtering to improve our results.

ACKNOWLEDGEMENT

This work was supported by JSPS KAKENHI Grant Number 16K16037 and 17H00731.

REFERENCES