Information and Software Technology 000 (2016) 1–21

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Search-based software library recommendation using multi-objective optimization

Ali Ouni[a,b,*], Raula Gaikovina Kula[a], Marouane Kessentini[c], Takashi Ishio[a], Daniel M. German[d], Katsuro Inoue[a]

[a] Department of Computer Science, IST, Osaka University, Osaka, Japan
[b] Department of Computer Science and Software Engineering, UAE University, UAE
[c] Department of Computer and Information Science, University of Michigan, MI, USA
[d] Department of Computer Science, University of Victoria, Victoria, Canada

ARTICLE INFO

Article history:
Received 2 December 2015
Revised 28 October 2016
Accepted 22 November 2016
Available online xxx

Keywords:
Search-based software engineering
Software library
Software reuse
Multi-objective optimization

ABSTRACT

Context: Software library reuse has significantly increased the productivity of software developers, reduced time-to-market and improved software quality and reusability. However, with the growing number of reusable software libraries in code repositories, finding and adopting a relevant software library becomes a fastidious and complex task for developers.

Objective: In this paper, we propose a novel approach called *LibFinder* to prevent missed reuse opportunities during software maintenance and evolution. The goal is to provide a decision support for developers to easily find "useful" third-party libraries to the implementation of their software systems.

Method: To this end, we used the non-dominated sorting genetic algorithm (NSGA-II), a multi-objective search-based algorithm, to find a trade-off between three objectives : 1) maximizing co-usage between a candidate library and the actual libraries used by a given system, 2) maximizing the semantic similarity between a candidate library and the source code of the system, and 3) minimizing the number of recommended libraries.

Results: We evaluated our approach on 6083 different libraries from Maven Central super repository that were used by 32,760 client systems obtained from Github super repository. Our results show that our approach outperforms three other existing search techniques and a state-of-the art approach, not based on heuristic search, and succeeds in recommending useful libraries at an accuracy score of 92%, precision of 51% and recall of 68%, while finding the best trade-off between the three considered objectives. Furthermore, we evaluate the usefulness of our approach in practice through an empirical study on two industrial Java systems with developers. Results show that the top 10 recommended libraries was rated by the original developers with an average of 3.25 out of 5.

Conclusion: This study suggests that (1) library usage history collected from different client systems and (2) library semantics/content embodied in library identifiers should be balanced together for an efficient library recommendation technique.

1. Introduction

Modern software systems build on a significant number of third-party software libraries to deliver feature-rich and high-quality software. Several studies have shown that software library reuse promotes efficient and effective software development. Consequently, library reuse leads to a significant increase in the productivity, reduction in time-to-market, improvement in the overall software quality, as well as reducing the inherent testing costs [1,2]. Reusing mature software modules can benefit from the collective experience of previous users of the module, as many bugs as well as deficiencies in the documentation have already been discovered [3,4].

Indeed, it is recognized that replacing legacy code with quality components and libraries typically reduces the amount of source code that must be maintained [5]. The benefits of replacing legacy

* Corresponding author.
E-mail address: ouniaali@gmail.com (A. Ouni).

http://dx.doi.org/10.1016/j.infsof.2016.11.007
0950-5849/

code by external quality software components was best articulated by Seacord et al. [5]: "*replacing functional components may also provide additional capabilities and improve on such attributes of system quality as robustness or performance*". In fact, replacing legacy code by third-party libraries has recently attracted much attention in both academia and industry. One example is the refactoring of "synchronized" blocks in Java by replacing them with the utility library `java.util.concurrent` [6,7].

Today, software systems utilize online code repositories such as Maven Central repository[1] to access to a host of reusable Open Source Software (OSS) libraries. Indeed, reusable software libraries are often reused multiple times and are therefore proven solutions that can provide better quality characteristics compared to newly developed code [8]. Recent empirical studies have found that 93.3% of modern OSS projects use third-party libraries, with an average of 28 libraries per project [9]. On the other hand, recent work indicate that developers are still often reinvent the wheel and spend effort and time, on re-implementing functionality, that could be saved by reusing mature and well-tested libraries [10,11].

We conjure two key reasons for this occurrence. First, due to the magnitude of available libraries we consider that, most of the time, developers are unaware or overwhelmed by related libraries. Online sources[2] report that available libraries are growing at an exponential rate. Hence, searching relevant software libraries can be a fastidious task for software developers, which would have an impact their productivity. Second, in addition to different reasons of distrust [12], developers are wary of the inherent costs and risks of library incompatibilities [13] associated with integrating new and unknown libraries into their existing systems. With the motto '*if not broke don't fix*', systems as a consequence risk outdated libraries.

To help developers, most of existing library recommendation approaches are based on commonly used together library methods, e.g., API usage patterns, at the method level of granularity [14–18]. The most related work of recommendation at the library level of granularity is by Thung et al. [9]. The authors use collaborative filtering and association rule mining on historic software artifacts to determine commonly used libraries without considering the library content. However, a library usage history-based approach would not be able to recommend libraries to projects that only use a small number of libraries or do not use any libraries at all. Thus, the content of a library is an extremely important asset that should be more informative and explicit for an effective library recommendation method. This approach deal with library recommendation as a single objective problem based on usage history. We believe that library recommendation is rather a complex decision making problem where several considerations should be balanced. These complex multi-objective decision problems with competing and conflicting constraints are well suited to Search Based Software Engineering (SBSE) [19,20].

To address the library recommendation problem, we introduce a novel approach called *LibFinder* based on the following two heuristics: a candidate library L can be useful for a given system S if (i) L has been commonly used with one or more libraries adopted by S, and (ii) L uses identical or similar identifiers, i.e., belongs to the same application domain, as S. To this end, we used the history of library usage as a 'wisdom of the crowd' and semantic similarity embodied in library identifiers mined from large code repositories from the internet. Our multi-objective formulation aims at finding optimal solutions providing the best trade-off between the three following objectives: 1) maximize co-usage between a candidate library and the actual libraries used by a given system, 2) maximize the semantic similarity between a candidate library's code and the system's code, and 3) minimize the total number of recommended libraries. To this end, we used the popular multi-objective search-based algorithm the non-dominated sorting genetic algorithm (NSGA-II) [21] to find the best trade-off between the three objectives. The complexity of the addressed library recommendation problem is combinatorial since our formulation consists of assigning libraries to different code fragments and the search is guided based on the above dependent evaluation functions.

To evaluate the efficiency of our approach, we used the history of 32,760 software projects mined from Github, that were clients for 6083 Maven libraries. The obtained results show that our approach is efficient in recommending relevant software libraries. We compare our approach with random search and two other popular search-based algorithms as well as a state-of-the-art approach. The statistical analysis shows better performance of our approach with 92% of accuracy, 51% of precision and 68% of recall. Furthermore, we evaluate the usefulness of our approach in practice through an empirical study on two industrial Java systems with developers. Results show that the top 10 recommended libraries was rated by the original developers of both systems with an average of 3.25 out of 5.

The main contributions of this paper can be summarized as follows:

1. We propose a new search-based approach called *LibFinder*, to detect and recommend third-party libraries that may be relevant to software systems that have already been implemented, and that it is intended for maintenance and evolution. To the best of our knowledge, this is the first attempt to use SBSE to address the library recommendation problem.
2. We collect a rich dataset by (i) mining the usage history, and (ii) extracting the identifiers of a large set of popular libraries from Maven Central Repository. The dataset is publicly available to encourage future research in the field of library recommendation[3].
3. We present an empirical evaluation of the performance of our approach using a 10-fold cross validation, along with statistical analysis of the obtained results. The obtained results show that our approach outperforms random search and two other search techniques at a confidence level of 95% and outperforms a state-of-the-art library recommendation approach [9] with an accuracy score of 92%, precision score of 51% and recall score of 68% while finding the best trade-off between the considered objectives. We present the results of a second empirical study to evaluate our approach in two industrial systems in real world setting where the recommended libraries were rated 3.25 out of 5 on average.

The rest of the paper is organized as follows. Section 2 presents the necessary background and a motivating example. Section 3 presents the basic concepts of our approach. Section 4 introduces our search-based approach for library recommendation *LibFinder*. Section 5 describes our empirical study and reports the obtained results, while Section 6 presents the threats to validity of the study. Section 7 presents the related work. Finally, Section 8 concludes and presents our future research directions.

2. Background and motivating example

In this section, we first describe the necessary background related to the proposed approach. We then present an example to help readers to better understand the motivation for library recommendation.

[1] http://search.maven.org.
[2] http://www.modulecounts.com, mvnrepository.com.
[3] http://sel.ist.osaka-u.ac.jp/~ali/libRecommendation/.

Please cite this article as: A. Ouni et al., Search-based software library recommendation using multi-objective optimization, Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.11.007