MigrationAdvisor: Recommending Library Migrations from Large-Scale Open-Source Data

Hao He*, Yulin Xu*, Xiao Cheng†, Guangtai Liang†, and Minghui Zhou*§
*Department of Computer Science and Technology, Peking University, Beijing, China
*Key Laboratory of High Confidence Software Technologies, Ministry of Education, China
Software Analysis Lab, Huawei Technologies Co., Ltd., China†
Email: {heh, kylinxyl, zhmh}@pku.edu.cn*, {chengxiao5, liangguangtai}@huawei.com†

***Abstract*—During software maintenance, developers may need to migrate an already in-use library to another library with similar functionalities. However, it is difficult to make the optimal migration decision with limited information, knowledge, or expertise. In this paper, we present MIGRATIONADVISOR, an evidence-based tool to recommend library migration targets through intelligent analysis upon a large number of GitHub repositories and Java libraries. The migration advisories are provided through a search engine style web service where developers can seek migration suggestions for a specific library. We conduct systematic evaluations on the correctness of results, and evaluate the usefulness of the tool by collecting usage feedback from industry developers. Video: https://youtu.be/4I75W22TqwQ.**

***Index Terms*—library migration, mining software repositories, library recommendation, dependency management**

I. INTRODUCTION

The wide adoption of open-source third-party libraries in modern software systems is beneficial but also risky. Third-party libraries may be abandoned by their maintainers, may have license incompatibilities, or may fail to satisfy new requirements due to absence of features, low performance, etc. To address these issues, developers need to migrate an already in-use library (i.e., *source library*) to another similar or functionality equivalent library (i.e., *target library*) for their software projects. Such activities are called as *library migration* in the related literature [1]–[4].

However, it is often non-trivial to find target libraries and choose the best target library for a software project. Existing library comparison information on the Internet (e.g., blog posts, forum discussions, community curated lists) are mostly opinion-based, likely outdated, and inherently controversial. Existing works on distilling library differences [5] and mining similar libraries [6] may help quickly locate a set of possible target libraries, but the former is still summarizing opinions and the latter provides no evidence on the feasibility of a migration. In practice, software projects rely heavily on core developers for making the migration decision, but the decision may be sub-optimal, especially when developers have limited knowledge or experience with the candidate target libraries.

To address this situation, several approaches have been proposed to mine existing library migrations from a large corpus of software repositories [1], [2], [4]. The underlying rationale is that historical migration practices provide valuable reference and guidance for developers when they make migration decisions. However, the existing works suffer from either low performance or limited scale which limits their usefulness in practice (More details in Section VI).

In this paper, we present MIGRATIONADVISOR, an accurate and evidence-supported library migration recommendation tool to support decision making before conducting a library migration in a Java project. Our tool works upon an advisory database built from a large corpus of Java GitHub repositories and Maven artifacts. Given a source library that a user wants to migrate for his/her project, it will generate a set of candidate target libraries from the repositories, compute a set of metrics for each candidate, and rank them using a combined confidence value. Finally, the ranked candidate targets and the possible migration commits for each target will be returned for human inspection. We expect our tool to be used by project maintainers to seek migration suggestions for a library they are already using, evaluate between a set of candidate target libraries, or support their migration decisions for a specific target library. We also plan to integrate our tool in an enterprise-level library management process which aims to support and secure third-party library usage in IT companies.

We systematically evaluate the correctness of recommendation results in a technical paper [7], showing that our tool can recommend migration target libraries with MRR of 0.8566, top-1 precision of 0.7947, top-10 NDCG of 0.7702, and top-20 recall of 0.8939. We further invite several industry developers to search libraries they know using our tool, and their feedback shows that the tool returns valid recommendations with useful evidences. They also provide interesting comments and insights for future research.

Our tool is available at http://migration-helper.net/. The source code, data, evaluation scripts, and a RESTful backend is available at https://github.com/hehao98/MigrationHelper. The source code for frontend is available at https://github.com/hehao98/MigrationHelperFrontend.

II. MIGRATIONADVISOR WORKFLOW

Figure 1 provides an overview of MIGRATIONADVISOR. It has two major components: a data preparation component and a data consumption component. The data preparation component aggregates and processes repository and library data from multiple sources, runs a recommendation algorithm

§corresponding author.

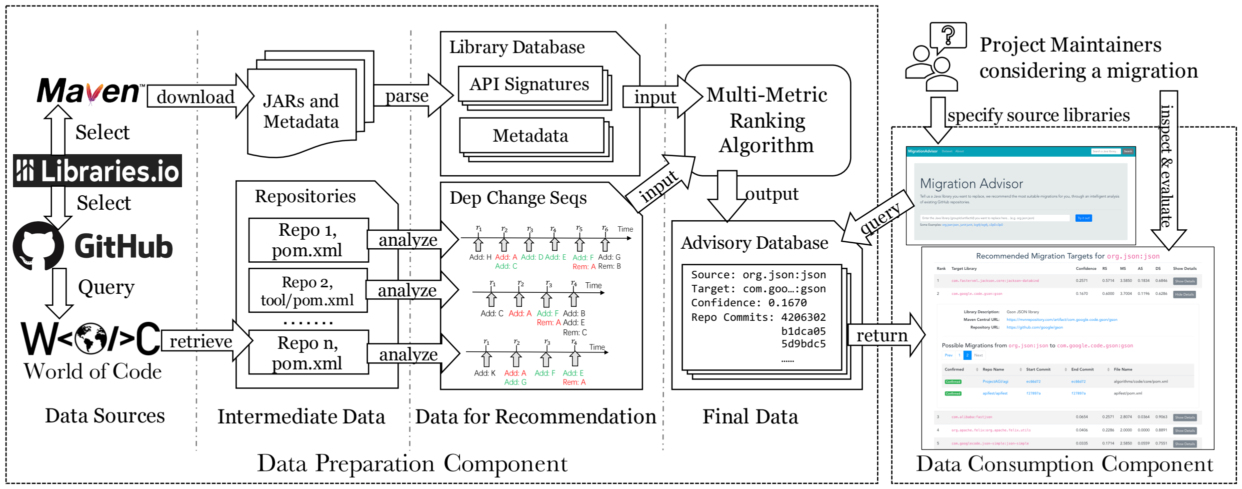


Fig. 1. Overview of MIGRATIONADVISOR, with two major components. The data preparation component aggregates library and repository data for generating the advisory database. The data consumption component is an interactive frontend where developers can search for migration advisories given a specific library.

for all libraries in the collected data, and stores the results in an advisory database. The data consumption component serves user requests when they seek migration advice for some given source libraries. Given a source library, it will query the advisory database and return a set of target libraries so that the users can inspect and evaluate different migration targets.

A. *Data Preparation*

Given an ecosystem we want to support, the data preparation component (Figure 1, left) aims to collect necessary information, evidence and existing migrations from the development histories of open-source projects. We choose to implement our tool for Java and Maven because of their popularity and industry importance. We use latest Libraries.io dataset [8] (last updated January 2020) for selecting libraries and repositories of interest, where we get a list of 184,817 distinct libraries (i.e. Maven artifacts with distinct group ID and artifact ID) and a list of 21,358 Maven managed non-fork GitHub repositories with at least 10 stars and three `pom.xml` changes.

For each library, we retrieve its version information and other metadata from Maven Central, resulting in 4,045,748 distinct library versions. For each version, we download its corresponding JAR file from Maven Central, if it has one, and extract all public classes from the JAR file. For each class, it is transformed into a compact API signature document which encodes all its public fields, methods, and inheritance relationships. The documents are stored in a library database. Other library metadata (e.g. versions, dependencies and descriptions) are also stored in the library database for further use.

For each GitHub repository, we retrieve all its version control data (i.e. commits, trees, blobs) from the World of Code database [9] (version R, last updated in April 2020), which includes all GitHub repositories mentioned before. We use World of Code because it offers much higher performance for analytical purposes, compared with directly cloning from GitHub and analyzing with `git`. We then collect all `pom.xml` files and all their historical versions in this repository. For each `pom.xml` file, we iterate over all its historical versions and compare with its previous version, to extract the dependency changes happened in this version. By merging dependency changes from different commits and branches, we generate a dependency change sequence for each `pom.xml` file, which contains all library adoption, removal and update histories for this `pom.xml` file.

Given the data described above, the final migration advisory database is generated by a multi-metric ranking algorithm for migration target recommendation. For each library we collect, the algorithm first generates a set of candidate target libraries by analyzing the dependency change sequences. Then, it computes several metrics for each candidate: Rule Support (RS), Message Support (MS), API Support (AS) and Distance Support (DS). The metrics are intended to identify real target libraries from the large number of false positives in the initial set of candidates, by capturing different dimensions of evidences from the collected data. More specifically, RS captures frequently added and removed libraries in the same commit; MS captures libraries that developers stated a migration in the commit messages; AS captures frequent code changes between the method calls of two libraries; and DS captures hints from commit topology. After that, it computes confidence value from all metrics using a simple multiplication

$$confidence(lib_A, lib_B) = RS \cdot MS \cdot AS \cdot DS \qquad (1)$$

which represents the likelihood that the candidate library is an eligible migration target for the source library. The algorithm is detailed in a technical paper [7].

To enhance user experience, we also mark a subset of ground truth migration advisories in the advisory database. The ground truth comes from a systematic labelling process detailed in [7], including source/target library pairs, migration commits, and related `pom.xml` changes. The ground truth migration advisories can also be extended and curated in a crowd-sourced manner when more users use our tool.

Table I shows the statistics of the data used in current tool implementation. The data can be periodically updated to reflect latest trends in the open-source community.

TABLE I
STATISTICS OF THE DATABASE USED IN OUR TOOL. API COUNT IS AN INTERMEDIATE DATA TABLE USED DURING METRIC COMPUTATION.

Data Type	Count or Size	Time to Construct
GitHub repositories	21,358	Several minutes
Commits with diffs	29,439,998	About 1 day
Parsed `pom.xmls`	10,009,952	About 1 day
Dep change seqs	147,220	Several hours
Libraries	185,817	Several minutes
Library versions	4,045,748	Several hours
Java classes	25,272,024	About 3 days
Non-zero API counts	4,934,677	About 2 weeks
Migration advisories	1,956,809	Several hours
Ground truth advisories	14,334	1 week (manually)
Database size (gzip dump)	∼50GB	Several weeks

B. Data Consumption

The data consumption component (Figure 1, right) serves as an interface to project developers and maintainers. Given a source library to be migrated, it should provide informative and interactive demonstrations of the migration recommendations from the advisory database. For current tool demo, it is implemented as a search engine style web service, where users can search for the migration targets of a specific library. Given a search query, the web service will retrieve all target libraries from the advisory database, sort them by the confidence value, and return them in a paginated table where each table entry demonstrates one target library. For each entry, users can also extend the entry for more detailed information, including library description, homepage, repository links, and GitHub repositories/commits that may have performed a migration from the given source library to the target library in this entry. Interested users can click on the links to browse more information about the target library and the migration commits.

III. IMPLEMENTATION

The data preparation component is implemented using a combination of Java programs and Python scripts, where each program or script implements one data processing stage (i.e. an arrow) in Figure 1. All programs and scripts are executed on one of the World of Code [9] server nodes, which is a Red Hat Linux server with 2 Intel Xeon E5-2630 v2 CPUs, 400GB RAM and 20TB storage. The total size of intermediate data (i.e., library JARs and raw git objects) exceeds 4 TB, but they are not needed for the data consumption component and can be safely deleted once the analysis is finished. The library APIs, library metadata, repository metadata, dependency change sequences, the final migration advisories, and other relevant data are stored in a local MongoDB instance (See Table I for its statistics). The data consumption component only needs to read from this database, so it is relatively lightweight and has flexible deployment options. Currently, the frontend, backend, and a MongoDB instance are hosted on an AWS virtual Linux server with 2 CPU cores, 8GB RAM, and 200GB storage.

IV. TOOL USAGE

We expect MIGRATIONADVISOR to be used when project maintainers discover that one of the libraries in their project need to be replaced (due to license, security, internal industry standards, etc). Suppose a developer wants to migrate from `org.json:json` because its license is incompatible with Apache 2.0 license. Using our web service, she can type `org.json:json` in the search input and click the "Search" button, as shown by the web page on the right part of Fig 1. Then the service will return a list of recommended migration targets, the top-3 being `jackson`, `gson` and `fastjson`. She can immediately discover that `jackson` seems to be a good choice because most projects migrate to `jackson` and it is also licensed under Apache 2.0. If she is more prudent, she can carefully investigate the migration commits and even combine our tool with other approaches to make the final decision.

In the future, we plan to integrate this component with existing third-party library checking software, either as an IDE plugin or as part of a CI/CD process. When the checking software identifies incompatible, deprecated, or banned libraries, our tool will prompt migration advisories to help developers quickly locate a target library that they can migrate to.

TABLE II
PERFORMANCE OF THE MULTI-METRIC RANKING ALGORITHM USED IN MIGRATIONADVISOR, COMPARED WITH OTHER EXISTING APPROACHES.

Approach	MRR	Precision@1	NDCG@10	Recall@20
Teyton et al.	0.7335	0.6757	0.6909	0.6391
Alrubaye et al.	0.9412	0.9412	0.9412	0.0540
Our Approach	0.8566	0.7947	0.7702	0.8939

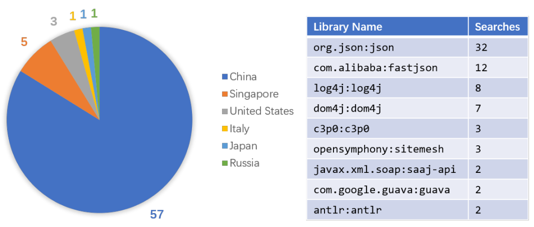
V. EVALUATION

Two aspects of MIGRATIONADVISOR are evaluated. The first aspect is the correctness and completeness of migration advisories (i.e. whether the returned results are real migration targets and whether all migration targets are returned). The second aspect is to what extent this tool is helpful for developers when they make migration decisions.

For the first aspect, we use the following common performance metrics for evaluating information retrieval and ranking problems: Mean Reciprocal Rank (MRR), top-k precision, top-k recall and top-k Normalized Discounted Cumulative Gain (NDCG). Table II shows the results of the algorithm used in our tool and the results of two existing approaches [1], [4] on the ground truth advisories. We can see from Table II that the multi-metric ranking algorithm significantly outperforms existing work, reaching MRR of 0.8566, top-1 precision of 0.7947, top-10 NDCG of 0.7702, and top-20 recall of 0.8939.

For the second aspect, we invite industry developers in our social network[1] to search libraries they know using our tool, and collect their usage feedback through informal communications. They provide positive feedback and interesting insights about our tool. One developer replies that *finding replaceable and actively maintained software is what we really need for legacy software*. They also suggest that our tool should return reasons for each recommendation, and issue warnings when no migration target is available. However, we also discover that only a small fraction of developers have

[1]This is done by sharing posts in a number of group chats (∼100 people in total), but it is hard to count the exact number of developers reached.



Library Name	Searches
org.json:json	32
com.alibaba:fastjson	12
log4j:log4j	8
dom4j:dom4j	7
c3p0:c3p0	3
opensymphony:sitemesh	3
javax.xml.soap:saaj-api	2
com.google.guava:guava	2
antlr:antlr	2

Fig. 2. User geographic information and most frequently searched libraries

experience on library migrations, and one reason may be that most developers we contact are in relatively junior positions and have limited years of working experience, but it should be explored in future research. By monitoring our website in a two-month period, we record search attempts from 68 different IPs. Figure 2 shows user geographic information and most frequently searched libraries.

VI. Related Work

Several existing tools and websites also aims to help developer select between libraries or conduct library migrations, but they all have limitations in this problem context. SimilarTech [6] recommends similar libraries through Stack Overflow tag embeddings, but it provide no evidence on the feasibility of migrations between the query library and returned libraries. LibComp [10] is a metric-based comparison tool for similar libraries, but it requires manual specification of two libraries and cannot help developers when they are unaware of a possible migration target. There are also community efforts such as AlternativeTo[2], a crowd-sourced software recommendation website, and `awesome-java`[3], a community curated Java library list organized into many categories. Library comparison blog posts and forum discussions[4] can be accessed using a search engine, and DiffTech [5] can be used to aggregate community opinions of similar libraries from online discussions. However, these approaches can only return opinion-based results which are inherently controversial and may not be trust-worthy. Despite their limitations, our tool is not intended to fully replace any of the existing approaches, but to *enhance* migration decision making through providing objective evidence of historical migration practices. Developers can refer to our tool and any existing approaches above to make the optimal decision for his/her project.

Some existing approaches share the same objective as our tool, but they suffer from performance issues which limit their usefulness in practice. Teyton et al. [1] propose a filtering-based approach on mining library migrations, but it suffers from either low precision or low recall depending on the filtering threshold. MigrationMiner [4] uses a different filtering-based approach, but it is only evaluated on 16 GitHub repositories (by contrast, 21,358 analyzed in our tool). To the best of our knowledge, our tool is the first accurate, large-scale, and evidence-supported library migration recommendation tool being able to recommend migration target libraries based on large-scale open-source data.

Other tools aim to support library migrations by providing API mappings (e.g. SimilarAPI [11]) or directly transform code to use the new library (e.g. Meditor [12]). Developers can use the results of our tool as the input to these tools, to improve development efficiency during library migrations.

[2]https://alternativeto.net/

[3]https://github.com/akullpp/awesome-java

[4]Unfortunately, the well-known Stack Overflow does not allow opinion-based discussions including the topic of library comparison and selection (See https://meta.stackoverflow.com/questions/255468/opinion-based-questions).

VII. Conclusion

We present MigrationAdvisor, an evidence-supported library migration recommendation tool which works upon a migration advisory database built from a large corpus of GitHub repositories and Java libraries. In the future, we plan to improve our tool from various perspectives, such as reason extraction, cost estimation, and customized recommendation.

Acknowledgment

This work is supported by the National Key R&D Program of China Grant 2018YFB1004201, the National Natural Science Foundation of China Grant 61825201, and the Open Fund of Science and Technology on Parallel and Distributed Processing Laboratory (PDL) No.6142110810403.

References

[1] C. Teyton, J.-R. Falleri, and X. Blanc, "Mining library migration graphs," in *2012 19th Working Conference on Reverse Engineering*. IEEE, 2012, pp. 289–298.

[2] C. Teyton, J.-R. Falleri, M. Palyart, and X. Blanc, "A study of library migrations in Java," *Journal of Software: Evolution and Process*, vol. 26, no. 11, pp. 1030–1052, 2014.

[3] S. Kabinna, C.-P. Bezemer, W. Shang, and A. E. Hassan, "Logging library migrations: A case study for the Apache software foundation projects," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 154–164.

[4] H. Alrubaye, M. W. Mkaouer, and A. Ouni, "MigrationMiner: An automated detection tool of third-party java library migration at the method level," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 414–417.

[5] H. Wang, C. Chen, Z. Xing, and J. Grundy, "DiffTech: A tool for differencing similar technologies from question-and-answer discussions," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1576–1580.

[6] C. Chen and Z. Xing, "SimilarTech: Automatically recommend analogical libraries across different programming languages," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 834–839.

[7] H. He, Y. Xu, Y. Ma, Y. Xu, G. Liang, and M. Zhou, "A multi-metric ranking approach for library migration recommendations," in *Proceedings of the 28th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2021.

[8] Libraries.io open data. [Online]. Available: https://libraries.io/data

[9] Y. Ma, C. Bogart, S. Amreen, R. Zaretzki, and A. Mockus, "World of code: An infrastructure for mining the universe of open source VCS data," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 143–154.

[10] R. El-Hajj and S. Nadi, "LibComp: An IntelliJ plugin for comparing java libraries," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1591–1595.

[11] C. Chen, "SimilarAPI: Mining analogical apis for library migration," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, 2020, pp. 37–40.

[12] S. Xu, Z. Dong, and N. Meng, "Meditor: Inference and application of API migration edits," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 335–346.