MigrationAdvisor: Recommending Library Migration Targets from GitHub Repositories

Hao He*†, Yulin Xu*, Xiao Cheng†, Guangtai Liang†, and Minghui Zhou*
School of Electronic Engineering and Computer Science, Peking University, Beijing, 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 used 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, a recommendation tool to objectively recommend library migration targets through intelligent analysis upon existing GitHub repositories. We have conducted systematic evaluations on the correctness of recommendations, and plan to evaluate the usefulness of our tool by collecting developer usage feedback in an industry context. An online introduction video is available at https://youtu.be/JZaVMWFfQO4 [[TODO: Replace with an English Video]].**

***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. For some libraries, a substitute may not exist at all; for other libraries, existing library comparison information (e.g. blog posts, forum discussions, community curated lists) are mostly opinion-based and likely outdated. 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 aggregating developer opinions and the latter provides no evidence on the feasibility, benefits, and cost of a migration from the source library to the target library. In practice, software projects rely heavily on the opinions of 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.

This work is supported by the National Key R&D Program of China Grant 2018YFB1004201 and the National Natural Science Foundation of China Grant 61825201.

To address this situation, several approaches has 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 work of Teyton et al. [1], [2] suffer from low performance and the work of Alrubaye et al. [4] is not tested on a large dataset (More details in Section V).

In this paper, we present MIGRATIONADVISOR, an accurate, scalable, 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 want 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 target candidates 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 industry third-party library management process which aims to support and secure third-party library usage in a large IT company.

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 0.7665, and top-20 recall of 0.8939. Our initial deployment in an industry context obtains positive feedback, and we plan to collect more developer feedback and refine our tool in the future.

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 provide 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

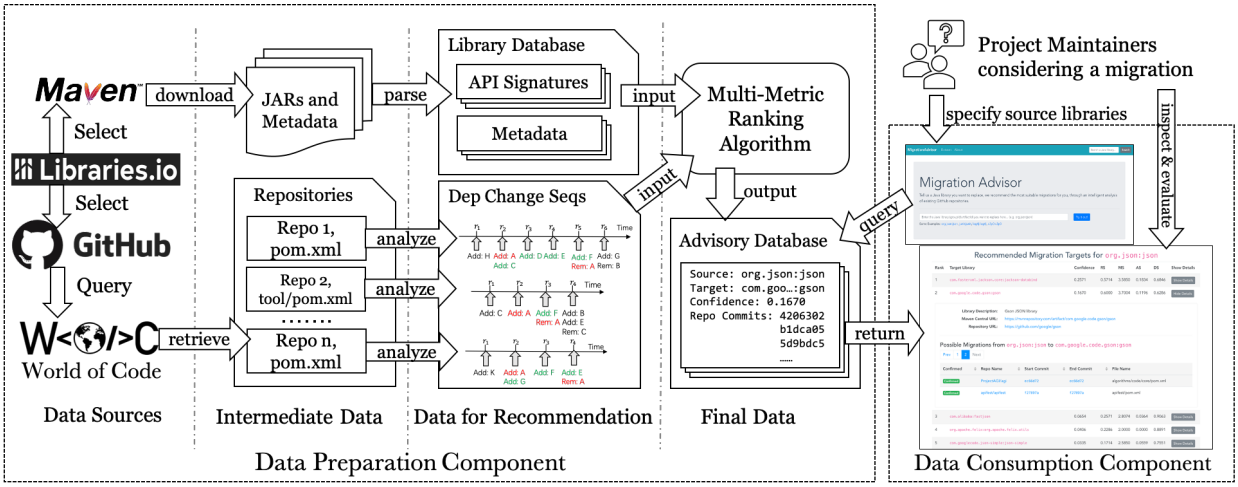

Fig. 1. Overview of MIGRATIONADVISOR.

data from multiple sources, run a recommendation algorithm for all libraries in the collected data, and store 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 a library ecosystem we want to support, the data preparation component aims to aggregate 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 in 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.

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 using Apache BCEL™. For each class, it is transformed into a compact API signature document which encodes all its public fields, methods, and inheritance relationships, and stored in a library database. The total size of downloaded JARs is ∼3TB, but they can be safely deleted after the parsing is finished. Other library metadata, such as 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], 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 its historical versions in this repository[1]. 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. Finally, all the sequences, along with other repository version control data, are stored in a database for further use.

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 collected, 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, Message Support, API Support and Distance Support. 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, Rule Support captures frequently added and removed libraries in the same commit; Message Support captures libraries that developers stated a migration in the commit messages; API Support captures frequent code changes between the method calls of two libraries; and Distance Support captures hints from commit topology. After that, the algorithm computes a confidence value from all metrics, which represents the probability that the candidate library is an eligible migration target for the source library. Finally, we store all source/target library pairs, metrics, confidence values, and related GitHub commits in the advisory database. The algorithm is detailed in a technical paper [7].

[1]In a Maven managed project, a `pom.xml` file declares all the libraries it use in the `<dependency>` section. A repository may have more than one `pom.xml` files in different paths, which are used to manage dependencies of different sub-modules or sub-projects.

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.xml`s	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	About 3 weeks

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.

B. Data Consumption

The data consumption component 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 also click on the links to browse more information about the target library and the migration commits.

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 some libraries that must be replaced (due to license, security, internal industry standards, etc), our tool will automatically prompt migration advisories to help developers quickly locate a target library that they can migrate to.

III. IMPLEMENTATION DETAILS

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. The statistics of this database are shown in Table I.

Once the data is ready, the data consumption component only needs to read from the MongoDB database in Table I, so it is relatively lightweight and has flexible deployment options. Currently, it is implemented as a demo web service, in which the backend is a RESTful Spring Boot application, and the frontend is a progressive web application built with Vue.js and Bootstrap. The frontend, backend, and a MongoDB instance are hosted on an AWS EC2 virtual Linux server with 2 CPU cores, 8GB RAM, and 200GB storage.

IV. EVALUATION

Two aspects of MIGRATIONADVISOR should be 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 is this tool helpful for developers when they make migration decisions. In this paper, we report the evaluation results of the first aspect (more details in the technical paper [7]) and a preliminary evaluation plan for the second aspect.

For the first aspect, we use the following common performance metrics for evaluating information retrieval and ranking problems: Mean Reciprocal Rank (MRR) [10], top-k precision, top-k recall and top-k Normalized Discounted Cumulative Gain (NDCG) [11]. The metrics are computed on a set of ground truth that we recover from an existing work [2] and systematically extend in latest GitHub repositories. Table II shows the results of the algorithm used in our tool and the results of two existing approaches [1], [4] for comparison. 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 0.7665, and top-20 recall of 0.8939 (Note that the the approach in Alrubaye et al. [4] has higher top-1 precision but its recall is significantly lower). Therefore, we conclude to the best of our knowledge that our algorithm achieves best overall performance, and our tool can provide the most accurate and complete migration advisories.

For the second aspect, we plan to invite developers to try our tool and fill in survey questionnaires using our website, not only to evaluate the current usefulness of our tool, but also to reveal future improvement and refinement directions. The questionnaire should contain questions about why their project need a migration, their concerns and considerations when selecting target libraries, and their opinions on our tool.

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	0.6952	0.6391
Alrubaye et al.	0.8461	0.8462	0.5440	0.0142
Our Approach	0.8566	0.7947	0.7665	0.8939

V. 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 the context of migration target recommendation. APIWave [12] tracks API popularity and migrations of 650 popular Java GitHub repositories, but its scale is limited and the most popular API changes in code may not be useful enough for developers seeking migration suggestions. SimilarTech [6] recommends similar libraries through Word2Vec embeddings trained from Stack Overflow tags, but it provide no evidence on the feasibility, benefits, and cost of a migration between the query library and returned libraries. There are also community efforts such as AlternativeTo [13], a website that collect alternatives to various technologies in a crowd-sourced manner, and `awesome-java` [14], a community curated Java library list organized into many categories. Library comparison blog posts and forum discussions[2] can be accessed using a search engine, and a recent approach [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 approaches mentioned above. Instead, the tool is intended to *enhance* migration decision making through providing objective evidence of historical migration practices, and developers can refer to our tool and any of the existing approaches above, in order to make the optimal migration 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 an filtering-based approach on mining library migrations, but it suffers from either low performance or low recall depending on the filtering threshold. MigrationMiner [4] uses a different 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, scalable, 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 [16]) or directly transform code to use the new library (e.g. Meditor [17]). Developers can use the results of our tool as the input to these tools, to improve development efficiency during library migrations.

[2]Unfortunately, the well-known Stack Overflow does not allow opinion-based discussions including the topic of library comparison and selection [15].

VI. CONCLUSION

In this paper, we present MIGRATIONADVISOR, an accurate and evidence-supported library migration target 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 further improve our tool by deploying in different contexts, collecting developer feedback, and uncovering practical facts about library migration.

ACKNOWLEDGMENT

We thank Professor Audris Mockus for providing computational resources, access to World of Code, and valuable usage instructions. We also thank our colleagues at Huawei for providing usage context and tool design suggestions.

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] Y. Huang, C. Chen, Z. Xing, T. Lin, and Y. Liu, "Tell them apart: Distilling technology differences from crowd-scale comparison discussions," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 214–224.

[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. (2020) A multi-metric ranking approach for library migration recommendations. [Online]. Available: https://hehao98.github.io/files/2021-migration.pdf

[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] N. Craswell, *Mean Reciprocal Rank*. Boston, MA: Springer US, 2009, pp. 1703–1703. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_488

[11] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, 2002.

[12] A. Hora and M. T. Valente, "APIWave: Keeping track of api popularity and migration," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 321–323.

[13] AlternativeTo: Crowd-sourced software recommendations. [Online]. Available: https://alternativeto.net/

[14] Awesome-Java: A curated list of awesome frameworks, libraries and software for the java programming language. [Online]. Available: https://github.com/akullpp/awesome-java

[15] Opinion based questions. [Online]. Available: https://meta.stackoverflow.com/questions/255468/opinion-based-questions

[16] 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.

[17] 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.