Towards Early Warning and Migration of High-Risk Dormant Open-Source Software Dependencies

Zijie Huang[†], Lizhi Cai[†, §], Xuan Mao[§, †], Kang Yang[†]
[†]*Shanghai Key Laboratory of Computer Software Testing & Evaluating, Shanghai Development Center of Computer Software Technology, Shanghai, China*
[§]*Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai, China*
{huangzj, clz}@sscenter.sh.cn, maoxuan@mail.ecust.edu.cn, yangkang@sscenter.sh.cn

***Abstract*—Dormant open-source software (OSS) dependencies are no longer maintained or actively developed, their related code components are more vulnerable and error-prone since they can hardly keep up with evolving software dependents. Presently, their migration remains costly and challenging for practitioners. To tackle such a challenge, we intend to characterize, predict, and automatically migrate high-risk dormant OSS dependencies. Our pilot study of 4,945 Maven dependencies reveals over half of them are dormant, and 12.15% pose a high security risk. These high-risk dependencies can be predicted early based on their version release and usage characteristics. They are rarely migrated by developers, and simple one-to-one API migrations can be achieved with little context using Large Language Models (LLMs). Future research will be conducted on a more complete dataset, incorporate socio-technical features for improved high-risk prediction, and fine-tune a migration code generator.**

***Index Terms*—dependency migration, open source sustainability, supply chain security, empirical software engineering**

I. Introduction

OSS dependencies facing sustainability issues may become dormant, i.e., no longer maintained or actively developed. Without updates, these dependencies become increasingly vulnerable, leading to maintenance and security challenges. Consequently, they are considered weak links [1] in the OSS supply chain and should be prioritized for migration.

However, migrating dormant dependencies involves several steps, each of which can be costly [2]. Developers must (1) identify the most critical and vulnerable dormant dependencies, (2) find suitable replacements, and (3) migrate every usage of the old library to the new one. As a result, this migration process can take years [3].

Recent research has explored the usage of dormant dependencies within the PyPI [4] and npm [3] ecosystems. Meanwhile, OSS dependency recommendation for migration [5], [6] is also an area of active study. However, to our knowledge, these studies do not adequately address the challenges posed by dormant OSS dependencies, as they offer only empirical insights or partial solutions for migration. Therefore, we aim to tackle this issue by predicting potential high-risk dormant OSS dependencies in advance, recommending suitable replacement dependencies, and automatically generating migration code.

The main contribution of this paper includes:

(1) We find that high-risk dormant OSS dependencies in Maven constitute 12.15% of all dependencies. These dependencies exhibit distinct characteristics regarding release intervals and usage patterns, and they are rarely migrated, regardless of the activity status of the associated projects.

(2) We present a preliminary prediction model for high-risk dormant OSS dependencies, alongside a case study demonstrating their migration using LLMs, showing the feasibility and challenges of building automated early warning and migration processes. The replication package is available in [7].

(3) We outline the future plans to improve the warning model and build a migration code generation model.

II. Pilot Study

In our pilot study, we aim to explore the necessity and feasibility of warning developers about potential high-risk dormant dependencies and facilitating their migration. We analyze the characteristics of high-risk dormant OSS dependencies, and investigate whether and how they are migrated. To ensure feasibility, we conduct preliminary predictions of such dependencies and perform a case study on their migration.

A. Characteristics of Vulnerable Dormant Dependencies

Data collection is conducted within the Maven ecosystem, as it has not been addressed in recent studies [3], [4], and it features widely utilized dependencies. We crawled the dependencies from category pages and the “popular” page between August 13 and 15, 2024. The collected information includes dependencies, vulnerabilities in the latest release, release timestamps for all versions, and usage categories for both the dependencies and their dependents. Dependencies without releases since January 1, 2022 are classified as dormant [8]. We excluded 0 usage dependencies, and we identified 2,637 dormant OSS dependencies out of a total of 4,945.

- **RQ1. Are dormant dependencies vulnerable?**

We collect data on the number and severity of CVEs (Common Vulnerabilities and Exposures) for dormant dependencies and their dependencies from mvnrepository [9] and the NVD API with Common Vulnerability Scoring System (CVSS) [10].

Finding 1. About half of the dormant dependencies are vulnerable, with a mean CVE count of 3.80, and an outlier CVE count threshold ($Q3 + 1.5IQR$ [11]) of 5.

- **RQ2. Why are dormant dependencies vulnerable?**

Only 1.09% of the 1,280 vulnerable dormant OSS dependencies are affected by direct CVEs, the majority are impacted by their dependencies, i.e., indicating transitive vulnerabilities.

Finding 2. Lacking maintenance of dependencies is the main reason that makes dormant dependencies vulnerable.

• **RQ3. Which dormant dependencies are riskier?**

We define a dormant dependency as high-risk if (1) it is affected by any CVE having critical or high CVSS severity, or (2) it has more than 5 CVEs, categorizing it as an outlier of CVE count based on Finding 1. Among the 1,280 vulnerable dormant dependencies, 46.95% are at high risk.

Since most vulnerabilities in dormant dependencies are introduced by their dependencies, we analyze their usage categories and scopes. The top 5 categories are presented in inset (a) of Fig. 1, while the distribution of the dependency scopes is shown in inset (b) of Fig. 1. In inset (c) of Fig. 1 we display the 10 most frequent vulnerable dependencies, labeled as `{scope}//{category}//{name}`.

Finding 3. Dependencies using utility, JSON, and I/O processing dependencies in the compilation scope are riskier. Test dependencies are also major sources of CVEs.

B. Migration Practice of High-Risk Dormant Dependencies

In this section, we outline the challenges posed by high-risk dormant dependencies and the urgency to tackle them.

• **RQ4. Are high-risk dormant dependencies migrated?**

To examine the presence of high-risk dormant dependencies and their impact on software projects, we collect projects affected by these dependencies. We identify their usage in `pom.xml` files using GitHub search and gather project information. We opt for GitHub search over its API due to the significantly lower number of results returned by API calls. Ultimately, we retrieved 7,997 affected projects. To our astonishment, none of them migrate dormant dependencies.

Finding 4. Preliminary studies indicate that dormant dependencies in the Maven ecosystem are rarely migrated.

• **RQ5. Do high-risk dormant dependencies affect active projects?**

To avoid drawing biased conclusions due to the lack of maintenance in the retrieved projects, we assess their development status and popularity. Among the 7,997 affected projects, 1,717 have more than 5 stars, and 37.68% of these 1,717 projects are active, i.e., have development activity since 2022.

Finding 5. More than one-third of the projects with high-risk dormant dependencies are actively maintained.

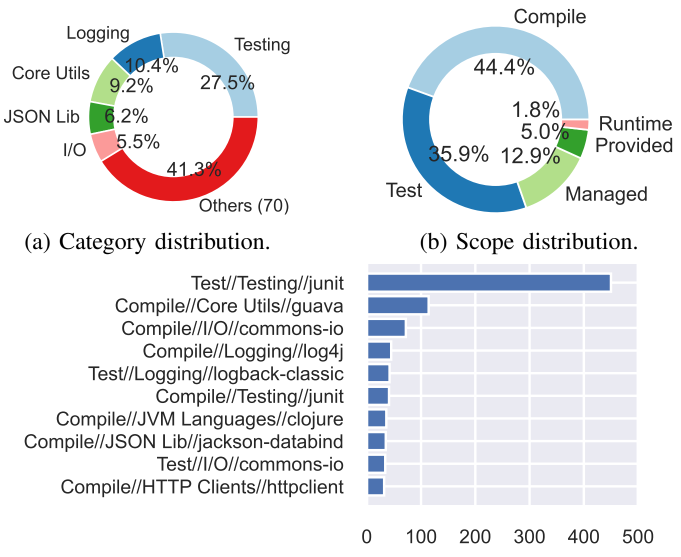
C. Feasibility Studies for Addressing Dormant Dependencies

In this section, we examine the feasibility of **warning** developers about high-risk dormant dependencies and **resolving** existing dormant dependencies by generating migration code under the condition that little migration presently occurs.

• **RQ6. Can dormancy and high-risk be warned early?**

1) Dormancy Prediction: We predict the dormancy of a dependency based on the timing of its latest release. To this end, we develop six predictive features: `days_since_prev_release`,`avg_days_release`, `days_since_first_release`,`num_releases_all_times`, `num_releases_last_year`, and `num_usage`. The first 5 features are calculated based solely on the day intervals between adjacent release dates, with intervals of less than one day being ignored. If a dependency has only 1 release, the first 5 features are set to 0. We use Random Forest as a classifier and employ shuffled stratified 10-fold cross-validation to evaluate its AUC, MCC, and F1 performance [12]. We achieve an ideal [13] mean AUC of 0.93, MCC of 0.72, and F1 score of 0.87. The boxplot summarizing the performance of each fold is presented in inset (a) of Fig. 2. The feature importance in Shapley values [14] indicate that `days_since_first_release` and `num_releases_all_times` are the most significant features that contribute to the model, with higher values correlating to a lower likelihood of dormancy. Conversely, a higher `days_since_prev_release` increases the prediction of dormancy. The feature importance figures and feature descriptions are available in the online appendix [7].

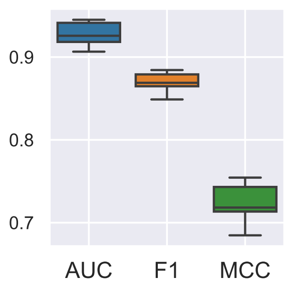
2) High-Risk Prediction: Using the previous dataset and similar experimental settings, we further predict high-risk dependencies among dormant dependencies. Given that transitive dependencies are a primary source of vulnerabilities, we develop 8 additional dependency features accordingly. These features are designed to (1) count the usage of dependencies in significant scopes, specifically `num_{compile,test}_scope_dependencies`, (2) count the usage of dependencies in important categories,



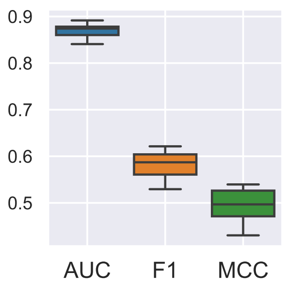
(a) Category distribution. (b) Scope distribution.

(c) Count of top-10 most frequent vulnerable dependencies.

Fig. 1: Statistics of vulnerable dependencies of high-risk dormant dependencies.



(a) Dormancy prediction.



(b) High-risk prediction.

Fig. 2: 10-fold cross-validation model performance boxplots.

namely `num_{testing,logging,core_utils,json_lib,io}_cat_dependencies`, and (3) count the number of all dependencies with `num_dependencies`. We achieve a mean AUC of 0.87, F1 score of 0.60, and MCC of 0.51, with the performance boxplot shown in inset (b) of Fig. 2. While there is room for improvement, especially for F1 score, we conclude that these features are valuable predictors.

Finding 6. Based on limited features of time intervals, usage categories, and scopes, dormancy can be well predicted, and the high-risk prediction model needs further improvement.

• **RQ7. Can dormant dependencies be migrated with minimal historical context provided?**

Recent studies consider library migration as a task that identifies equivalent API mappings between 2 libraries based on migration histories [5], [15]. However, the findings from RQ6 indicate that historical migration may not exist. Therefore, we investigate whether LLMs such as CodeLlama-70b and GPT-4o-mini can facilitate such migrations, since LLMs perform well in tasks requiring understanding API relations and generating unseen code, e.g., code translation [16]. The input prompt is: "This class uses the vulnerable {target library} as a dependency, migrate the {target library} usages to {destination library} to {purpose}, the code is as follows: {code}."

First, we use `net.sourceforge.jexcelapi:jxl`'s usage [17] as an example, its purpose is `processing Excel files`. The `jxl` library became dormant in 2011 and is categorized as an "Excel Library." `Apache POI` is the most popular active Excel library, so we migrate `jxl` to `POI`. Both libraries have one-to-one API mappings. GPT-4o-mini produces correct output that preserves the original behavior, while CodeLlama-70b generates API calls in the wrong package, requiring slight modifications for compatibility.

Next, we generate migrations from `com.github.pmerienne:trident-ml` to `WEKA` [18]. The `trident-ml` dependency is a `machine learning` library that became dormant in 2015, while `WEKA` is a well-known active alternative. This migration has more complex one-to-many API mappings in TF-IDF calculation for a cosine similarity evaluation script [19], and the LLMs fail to produce executable results.

In conclusion, the primary failure reasons are (1) complex one-to-many API mappings, (2) hallucinations causing the generation of inexistent API calls (inlining with [20], [21]), and (3) producing empty functions requiring implementation. The results can be found in the online appendix [7].

Finding 7. In our case study, LLMs could migrate dependencies having one-to-one API mappings with minimal context, but failed in more complex one-to-many API mappings.

III. Future Plans

Fig. 3 provides an overview of our future plans. Building on the findings from our pilot study, we aim to address RQ1-7 in a more rigorous and comprehensive manner by conducting an in-depth characterization of high-risk OSS dormant dependencies, issuing warnings about potential high-risk dependencies, and migrating them using context-enhanced LLMs. These proposed approaches will be applied and tested on industrial Java web-apps in government and fin-tech sectors.

A. Socio-Technical Characterization of High-Risk OSS Dormant Dependencies

1) Motivation: Our goal is to **answer RQ1-5 more effectively** by understanding the socio-technical characteristics of widely used, highly vulnerable, and difficult-to-migrate high-risk dormant dependencies. While our pilot studies in RQ1-3 focus on the surface characteristics of dormant dependencies, not all of these dependencies are high-risk; some may have transitioned to dormancy after achieving their primary development goals. To enhance prediction accuracy, we aim to incorporate more socio-technical features [22] that capture the health and functionality of dependencies. Pilot studies in RQ4-5 examine the impact of dormant dependencies within a limited set of projects, and we plan to expand this scope to validate our findings across a broader range.

2) Approach: First, we identify high-usage dependencies and the affected repositories from World of Code (WoC) [23], a software supply chain analysis infrastructure containing complete mirrors of open source software repositories such as GitHub. We extract both technical and socio-technical features from the affected repositories to characterize high-risk dormant dependencies. The technical information mainly includes details related to dependencies and their applications, such as API usage, code semantics, and code quality. In contrast, the socio-technical information considers factors related to the updates and maintenance of dependencies, including code turnover rates, developer retention rates, the number of pull requests and code reviews, commit intervals, the duration of development, and the network density, distance, and accessibility features of the development community. These factors can significantly influence the maintenance status of dependencies.

Next, we aim to establish improved rules for identifying high-risk dormant dependencies. The identification criteria may include significant slowdowns in updates, cessation of maintenance, marking as archived, and the latest version being labeled as outdated. High-risk identification rules might involve the presence of severe CVEs in the latest version, the number of CVEs exceeding a specific threshold, and the count

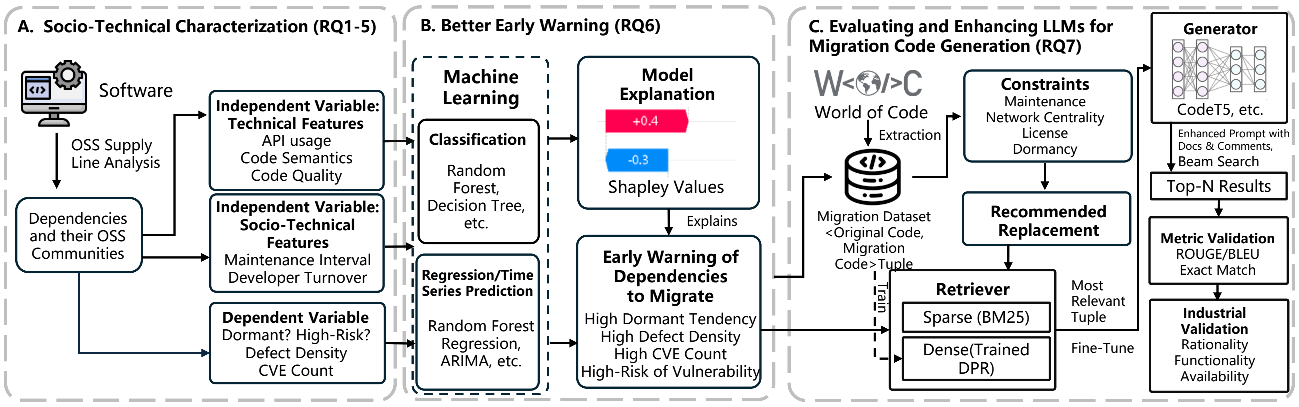


Fig. 3: An overview of future plans.

of unresolved or unassigned defect reports. We will invite professional developers to validate these rules and the findings.

B. Better Warning of High-Risk OSS Dormant Dependencies

1) Motivation: Our goal is to **address RQ6** by warning developers about unhealthy dependencies early, i.e., **knowing what and when to migrate**. Our pilot study in RQ6 indicates that warnings can be issued at the time of the last release. We aim to enhance this preliminary model by incorporating the socio-technical features developed in the previous subsection.

2) Approach: First, we extract usage features and socio-technical characteristics as independent variables for prediction. As dependent variables, we use high-risk, dormancy, and the number of severe defects [24] and CVEs as targets, incorporating both security and quality aspects.

Second, we approach the prediction of dormancy and high risk as a classification problem, while treating the prediction of severe defects and vulnerability counts as either a regression or time series forecasting problem. The models will be validated using time-wise [25] approaches and will produce predicted probabilities for dormancy and high risk, along with estimates for the number of defects and vulnerabilities. Additionally, SHAP [14] will be utilized to calculate feature importance based on model predictions and to explain model behavior.

C. Evaluating and Enhancing LLMs for OSS Dormant Dependency Migration Code Generation

1) Motivation: Our goal is to **address RQ7** by generating code for migration, i.e., **knowing migrate to which library and how to achieve it**. Based on Finding 7, this section primarily focuses on developing a state-of-the-art approach for one-to-one API migrations, tackling challenges such as hallucinations and responsiveness issues by retrieval augmentations and fine-tuning [21]. With a more robust model, we will explore solutions for more complex API mappings.

2) Approach: First, we recommend an alternative dependency. We plan to build a migration dataset using WoC. Then, all available migration targets will undergo filtering based on multiple constraints, including maintenance quality [26], centrality in the dependency network [27], licenses [28], and dormancy tendencies. Maintenance quality considers factors such as whether the library implements security measures recommended by OpenSSF [29] or GitHub, as well as the speed of vulnerability resolution. Network centrality constraint filters out less popular dependencies in the OSS ecosystem by measuring their usage. The license compliance constraint ensures the recommended open-source dependencies do not conflict with the target project's license. The dormancy tendency constraint exploits the prediction methods from the previous section to avoid recommending poorly maintained dependencies. Then, the remaining most frequently migrated libraries will be treated as candidates; if no migration history is available, we will select the most popular library that meets the constraints as an alternative.

Second, we train and fine-tune the retrieval and the code generation model. The migration code will be treated as a patch for the original code, using both the code to be migrated and the migration code as context for enhanced retrieval context. We will train a retriever and fine-tune an open-source pre-trained code model (e.g., CodeT5 [30]) as the migration code generator. Both dense and sparse retrieval algorithms will be employed to construct the retriever, with the training process primarily focusing on the dense retriever. The output from the retriever will be used to fine-tune the pre-trained model, enhancing its understanding of code migration tasks.

Third, we evaluate our model alongside existing alternatives. We use prompts that outline the migration's needs while incorporating available knowledge from documentation and comments to enhance the context, and select the most relevant result through beam search. We will create multiple datasets of varying difficulty [31], e.g., simple datasets for one-to-one mappings with more context and challenging datasets for one-to-many or many-to-one mappings, and progressively enhance performance on the more difficult tasks. The outcomes will be validated by both developer feedback and quantitative metrics.

IV. Conclusion

We conducted a pilot study to predict high-risk dormant OSS dependencies based on their usage characteristics and release intervals, while also assessing their migration status and feasibility. Results indicate developers seldom migrate them, our prediction model demonstrates ideal performance, and the potential of LLMs for migration is also highlighted.

Based on these observations, we proposed future plans including an in-depth socio-technical analysis of high-risk dormant OSS dependencies, the development of a prediction model that incorporates additional quality and security aspects, and a context-augmented migration code generation model.

Acknowledgment

This work was supported in part by the China Postdoctoral Science Foundation (2024M761927), the Shanghai "Science and Technology Innovation Action Plan" Star Cultivation Program (Sailing Program, 24YF2719900, 24YF2720000), and the Science and Technology Project of State Grid Hebei Energy Technology Service Co., Ltd., China (TSS2020-10).

References

[1] N. Zahan, T. Zimmermann, P. Godefroid, B. Murphy, C. Maddila, and L. Williams, "What are weak links in the npm supply chain?," in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '22, pp. 331–340, 2022.

[2] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, "Vulnerable open source dependencies: Counting those that matter," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, pp. 42:1–10, 2018.

[3] C. Miller, M. Jahanshahi, A. Mockus, B. Vasilescu, and C. Kästner, "Understanding the response to open-source dependency abandonment in the npm ecosystem," in *Proceedings of the 47th International Conference on Software Engineering*, ICSE '25, to Appear, 2025. https://tinyurl.com/d6w95tmh.

[4] Z. Zhong, S. He, H. Wang, B. Yu, H. Yang, and P. He, "An empirical study on package-level deprecation in Python ecosystem," in *Proceedings of the 47th International Conference on Software Engineering*, ICSE '25, 2025, https://arxiv.org/abs/2408.10327.

[5] 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 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering*, SANER' 21, pp. 72–83, 2021.

[6] J. Zhang, Q. Luo, and P. Wu, "A multi-metric ranking with label correlations approach for library migration recommendations," in *Proceedings of the 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering*, SANER' 24, pp. 183–191, 2024.

[7] Anonymous, *"Replication package (anonymous private link)"*, 2024, https://figshare.com/s/a9e38b571cef89552078.

[8] C. Miller, C. Kästner, and B. Vasilescu, ""We feel like we're winging it:" A study on navigating open-source dependency abandonment," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE '23, pp. 1281–1293, 2023.

[9] Mvnrepository, "Maven repository," 2024, https://mvnrepository.com.

[10] NVD, "NVD APIs," 2024, https://nvd.nist.gov/developers/vulnerabilities.

[11] M. Dilhara, A. Ketkar, and D. Dig, "Understanding software-2.0: A study of machine learning library usage and evolution," *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 4, pp. 55:1–42, 2021.

[12] J. Yao and M. Shepperd, "Assessing software defection prediction performance: Why using the matthews correlation coefficient matters," in *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, EASE '20, pp. 120–129, 2020.

[13] Z. Huang, H. Yu, G. Fan, Z. Shao, Z. Zhou, and M. Li, "On the effectiveness of developer features in code smell prioritization: A replication study," *Journal of Systems and Software*, vol. 210, p. 111968, 2024.

[14] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 4768–4777, 2017.

[15] C. Teyton, J.-R. Falleri, and X. Blanc, "Automatic discovery of function mappings between similar libraries," in *Proceedings of the 20th Working Conference on Reverse Engineering*, WCRE '13, pp. 192–201, 2013.

[16] Z. Yang, F. Liu, Z. Yu, J. W. Keung, J. Li, S. Liu, Y. Hong, X. Ma, Z. Jin, and G. Li, "Exploring and unleashing the power of large language models in automated code translation," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 71:1–24, 2024.

[17] GitHub, "Jxl usage," 2024, https://tinyurl.com/5n7wt7ps.

[18] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explorations Newsletter*, vol. 11, pp. 10–18, Nov. 2009.

[19] GitHub, "trident-ml usage," 2024, https://tinyurl.com/2ezka7cf.

[20] J. Latendresse, S. Khatoonabadi, A. Abdellatif, and E. Shihab, "Is ChatGPT a good software librarian? An exploratory study on the use of ChatGPT for software library recommendations," 2024, https://arxiv.org/abs/2408.05128.

[21] J. Spracklen, R. Wijewickrama, A. H. M. N. Sakib, A. Maiti, B. Viswanath, and M. Jadliwala, "We have a package for you! A comprehensive analysis of package hallucinations by code generating llms," 2024, https://arxiv.org/abs/2406.10279v2.

[22] F. Palomba and D. A. Tamburri, "Predicting the emergence of community smells using socio-technical metrics: A machine-learning approach," *Journal of Systems and Software*, vol. 171, p. 110847, 2021.

[23] Y. Ma, T. Dey, C. Bogart, S. Amreen, M. Valiev, A. Tutko, D. Kennard, R. Zaretzki, and A. Mockus, "World of code: Enabling a research workflow for mining and analyzing the universe of open source vcs data," *Empirical Software Engineering*, vol. 26, no. 2, pp. 1–42, 2021.

[24] X. Yu, J. Keung, Y. Xiao, S. Feng, F. Li, and H. Dai, "Predicting the precise number of software defects: Are we there yet?," *Information and Software Technology*, vol. 146, p. 106847, 2022.

[25] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung, "Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '16, pp. 157–168, 2016.

[26] A. H. Ghapanchi, "Predicting software future sustainability: A longitudinal perspective," *Information Systems*, vol. 49, pp. 40–51, 2015.

[27] L. Hossain and D. Zhou, "Measuring OSS quality trough centrality," in *Proceedings of the 2008 International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '08, pp. 65–68, 2008.

[28] W. Xu, H. He, K. Gao, and M. Zhou, "Understanding and remediating open-source license incompatibilities in the PyPI ecosystem," in *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering*, ASE '23, pp. 178–190, 2024.

[29] W. Enck and L. Williams, "Top five challenges in software supply chain security: Observations from 30 industry and government organizations," *IEEE Security & Privacy*, vol. 20, no. 2, pp. 96–100, 2022.

[30] Y. Wang, W. Wang, S. Joty, and H. C. Steven, "CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, EMNLP '21, pp. 8696–8708, 2021.

[31] Z. Zhou, H. Yu, G. Fan, Z. Huang, K. Yang, and J. Zhang, "HQLgen: deep learning based HQL query generation from program context," *Automated Software Engineering*, vol. 29, no. 2, p. 55, 2022.