An Empirical Study on the Evolution and Maintenance of the Maven Central Dependency Graph

Nirajan Bhattarai[1], Riley Stratton[2], Minhaz Zibran[2*]
[1]*Department of Biomedical and Pharmaceutical Sciences*, *Idaho State University*, Pocatello, USA
nirajanbhattarai@isu.edu
[2]*Department of Computer Science*, *Idaho State University*, Pocatello, USA
rileystratton@isu.edu, [*]minhazzibran@isu.edu

***Abstract*—Frequent new releases are expected to address security and compatibility issues, but may result in security breakages in return, and increase maintenance cost and system complexity. The primary aim of this paper is to conduct an empirical study on the ecosystem-wide evolution of the Maven Central Dependency Graph and the maintenance status of its artifacts, with the ultimate goal of educating the developer community about the evolution and maintenance levels within the Maven Central ecosystem. We examined the growth patterns of library releases over time, focusing on frequency, rhythm, and speed, to understand the influence of project management methodologies such as Agile, security vulnerabilities, and open-source contributions on the evolution of the Maven Central ecosystem. Similarly, we also attempted to measure the maintenance level of artifacts in the ecosystem based on various factors such as popularity, missed releases, and whether vulnerabilities have been addressed or not. Rapid growth between 2004 and 2005 aligned with emergence with agile methodologies, followed by a gradual increase up to 2021, after which it started plateauing or stabilizing, which may be attributed to the emergence of new technologies like AI. In contrast, the growth of vulnerabilities across all severity types experienced a rapid decline after 2021, possibly indicating an increased emphasis on security and improved library maintenance within the community. We found that 5.21% and 94.74% of all libraries are fully and partially maintained, respectively. The study suggests the need for further in-depth research on the most prevalent vulnerabilities in the Maven Central ecosystem, and a well-validated metric system for assessment of combined vulnerability and maintenance level in the Maven Central Dependency Graph.**

***Index Terms*—Maven Central, Dependency Graph, Evolution, Maintenance,Release, Vulnerability, Dependency**

I. Introduction

Dependencies are third-party software components introduced into the software development framework to reduce costs and accelerate the speed of software development. As dependencies are an indispensable component of the software development process, frequent new releases are expected to address security and compatibility issues, ensuring safe and reliable use [1]. However, maintenance effort on these dependency libraries incurs costs and increases complexity [2]. Raemaekers et al. highlighted that there will always be tension between version updates intended for mitigating security risks and the security breakages that may result from such updates [3].The study on the evolution of dependency can be approached from two perspectives: ego-centric, focusing on particular dependency libraries; and/or ecosystem-wide, having global insight on dependency ecosystem management and pervasion [4], [2].To assist with these kinds of studies, several repositories and resources are available, such as Maven Central, which maintains over 2.8 million unique artifact repositories for JVM-based artifacts, along with metadata detailing versions, dependencies, and upload dates in a dependency graph. This makes it well-suited for both ego-centric and system-wide big data evolution studies [5]. Similarly, various validated metrics have been proposed and incorporated in the Maven Central Database for researchers, eliminating the need to develop and validate their own quantification metrics, which would be particularly challenging for a dataset of this size. Some of these proposed metrics are popularity [6], calculated based on the number of dependents a particular version of a library has in a year, and the freshness [1], which is the computation of the number of releases and the duration elapsed between a particular release and its most recent update.

The primary aim of this paper is an ecosystem-wide empirical study of the evolution of the Maven Central Dependency Graph and the maintenance status of its artifacts. The evolution was assessed in terms of the growth of software artifacts, their releases, associated dependencies, and vulnerabilities of varying severity. Additionally, we quantified the extent of maintained and unmaintained libraries across the entire ecosystem based on the criteria we developed. The ultimate goal is to inform the developer community about the evolution and maintenance levels in the Maven Central Ecosystem, which may indicate the impact of different dynamics in the software engineering field, such as the evolution of agile development methodologies.

II. Method

A. Research Questions

Q1.What are the patterns in the growth of the Maven Central graph across different periods? The evolution of the ecosystem, especially within the Java community, was assessed by analyzing the pattern in the growth of the Maven

central graph across different periods. Assessing the frequency, rhythm, and speed trend of library releases over time can shed light on the developer's behavior and the impact of project management methodologies like Agile [7], [6]. Similarly, approximately 35% of all open-source projects have their origination from dependent libraries, which are increasingly susceptible to security vulnerabilities [8].

Q2.To what extent does the ecosystem contain unmaintained libraries? In the modern software development realm, open-source projects play an indispensable part. At the same time, open access has also remained largely unmaintained [9], mainly because they are maintained on the developer's unrestricted schedule and mostly unpaid [10]. Although open-source projects will likely remain unmaintained, limited studies address this issue [9].

B. Data Acquisition and Pre-processing

For data acquisition, we selected `with_metrics_goblin_maven_30_08_24 .dump`, which included various metrics, and was queried to extract all the information along with metrics associated with artifacts, dependencies, and releases. The dataset is provided as part of the 2025 Mining Software Repositories (MSR) Mining Challenges [4].Each release node has a `Timestamp` attribute, representing the exact moment the release occurred, stored in milliseconds since the Unix epoch (January 1, 1970). The extraction of the year for each release was carried out after formatting the timestamp into a year using the APOC date formatting function in Cypher (`apoc.date.format`). The number of libraries and releases per year was then plotted on a line graph to visualize the trend of library releases over time. The function to convert the timestamp (milliseconds since the Unix epoch) to a human-readable year is as follows: `year=convertMillisecondsToDate (timestamp).getYear()`

C. Metrics

1) Growth: The growth of all releases, all artifacts, and the prevalence of vulnerabilities of different severity types within a particular year were presented as a year-wise frequency trend in a line graph. The dependency count across years was presented as a box plot.

2) Release rhythm: A release rhythm can be defined as a sequence of time intervals (measured in days) between consecutive releases of an artifact, x_a, within a specific year, y_b [2]. It can be mathematically expressed as follows:

$$R_{|x_a,y_b|} = \{\Delta t_1, \Delta t_2, \ldots, \Delta t_{N_{a,b}-1}\} \quad (1)$$

$R_{|x_a,y_b|}$ represents the release rhythm of an artifact, x_a, within a specific year, y_b. y_b, and $n_{a,b-1}$ represents the number of releases. Δt represents the difference in days between consecutive releases of an artifact.

3) Release Speed: The speed of release of an artifact can be defined as the number of releases of an artifact, x_a, within a specific year, y_b, per total days from its first release to the last release in that year [2].

$$\text{Speed}_{|x_a,y_b|} = \frac{N_{a,b}}{\Delta T_{a,b}} \quad (2)$$

where $\Delta T_{a,b}$ represents the time interval (measured in days) between the first release and the last release of an artifact, $N_{a,b}$ represents the number of releases of artifact x_a within a specific year y_b, and $N_{a,b-1}$ represents the number of releases in the previous year.

4) CVE count and total vulnerability contribution: CVE is presented in JSON format for each release, with key "cve", which represents array of CVE identifiers. The JSON format is first cleaned, and subjected to count, which is measured as the number of the elements in the array. Similarly, the count of each severity level, where were 'LOW', 'MEDIUM', 'HIGH', 'CRITICAL', and 'NONE', within the CVE for each release was extracted. Since each release may have multiple CVE with different severity levels, we calculated the total vulnerability contribution score based upon severity level of vulnerabilities along with their count in respective severity. The total vulnerability contribution for each release is computed by multiplying the weight assigned to each severity level by its count and then summing them up, as follows: The total vulnerability contribution is calculated as follows:
Total vulnerability contribution =COUNT('LOW') X 3.0 + COUNT('MEDIUM') X 5.5 + COUNT('HIGH') X 8.0 + COUNT('CRITICAL') X 9.5 + COUNT('None') X 0.
For the release having no vulnerabilities, total vulnerability contribution was assigned as zero. The choice of weights is based on the vulnerability severity scale [11].

D. Maintenance level

Since there is no single available criterion to classify a software artifact as well-maintained based upon available metrics in the data, we devised our own criteria based on logical steps based upon different related metrics. Popularity, total vulnerability contribution, and the number of releases were primarily used to determine the maintenance level of the artifact. The classification was based upon three criteria: 1) whether its total vulnerability contribution has decreased at least two times or remained consistently below score of 8, which is threshold for high vulnerability; 2) whether the artifact's popularity has increased at least two times; 3) whether at least one release occurred within the last six months. Artifact having all three criteria fulfilled are classified as 'Fully Maintained,' those meeting two criteria as 'Partially Maintained,' those meeting one criterion as 'Slightly Maintained,' and those meeting none as 'Not Maintained'.

III. Results

A. Growth in Maven Central Ecosystem

1) Year-wise growth in software artifacts, its releases and dependencies: The trends show continuous growth of software artifacts and their releases over the years. (Figure 1, left) In particular, rapid incremental growth was observed from 2004 to 2005, when the scaling approach to agile software development was adopted. From 2005 to 2021, the growth was incremental but gradual. After 2021, where emerging

technologies like artificial intelligence started to manifest into the software development domain, the growth began to plateau. Regarding the growth of dependencies from the box plot, the growth, assessed by the median number of dependencies each year, shows an increase after 2011, where DevOps and Embedded systems adoption started, with a slight rise in 2020, a decrease in 2021, and a return to the previous growth trend beyond 2021.

2) Speed and rhythm of software releases in the ecosystem: The results on rhythm (Fig. 2) indicate that the timeline can be divided into five phases: Phase I (2002–2005), Phase II (2006–2010), Phase III (2009–2010), Phase IV (2011–2016), and Phase V (2017–2024). The duration between consecutive library releases of an artifact was subdued during Phase I, showed a ripple effect (i.e., increases and decreases) in Phases II and III, decreased and stabilized in Phase IV, and slightly decreased and stabilized further in Phase V. Phase I represents the period between integration and scaling, Phase II marks the transition from scaling to game design, Phase III corresponds to the emergence and adoption of DevOps practices, Phase IV reflects the period following the emergence of microservices, and Phase V encompasses the era of emerging technologies. Similarly, there has been steady growth in the speed of releases across the years within the entire ecosystem.

3) Trends and distributions of CVEs and their severity in the ecosystem: The emergence of new technologies, such as AI and automation in agile methodologies, may have contributed to this decline, along with growing concerns about security in software engineering in recent years (Figure 3, left). Releases with low and medium severity dominate the ecosystem, covering approximately 80% of all releases. However, highly vulnerable releases account for 14%, with moderate and high vulnerabilities collectively representing a significant portion of the ecosystem. Importantly, the top five most prevalent vulnerabilities were of high severity, ranging from moderate to critical (Table 1).

CVE ID	Severity	# Occurrence
CVE-2023-24057	CRITICAL	1138
CVE-2022-31159	HIGH	1420
CVE-2023-32070	CRITICAL	1691
CVE-2023-6835	MODERATE	1817
CVE-2023-6837	HIGH	3916

TABLE I
THE TOP FIVE MOST FREQUENT VULNERABILITIES, PRESENTED AS CVE IDS WITH THEIR SEVERITY LEVELS, OCCURRENCES, AND REFERENCES FOR DETAILED INFORMATION.

B. Maintenance level in the ecosystem

Based on the criteria we set for determination of the maintenance level of an artifact, we found that artifacts fulfilling two out of three criteria (partially maintained) were significantly more prevalent, followed by artifacts meeting all three criteria (fully maintained). Surprisingly, the number of artifacts meeting only one of the three criteria was significantly smaller. This may be because most artifacts already satisfied two or three of the criteria, we established for identifying maintenance levels, leaving a small proportion of artifacts that only met one of the three criteria (Figure 4). The percentage of partially maintained, fully maintained and slightly maintained libraries were 94.74%, 5.21%, and 0.05% respectively.

IV. DISCUSSION

In this study, we studied evolution all three components of software eco-system: software libraries, its releases, and dependences of releases across years, with special emphasis on rhythm, speed, maintenance and vulnerabilities. We observed growth over the years in terms of frequency, speed, and vulnerabilities until 2021. A study by Decan et al. also had similar observation, increase in dependency network over time [13]. Similarly, observing the rhythm of an artifact's lifecycle provides valuable insights into patterns and release speeds, enabling meaningful comparisons [2] . We analyzed the evolution in relation to the development of various agile methodologies to identify any significant correlations or associations.

After 2021,the growth began to plateau, with a decline in the growth of vulnerabilities. Beyond 2021, the period is marked with emergence of artificial intelligence and secure system design in the software development practices. The integration of artificial intelligence with Agile DevOps has promoted adaptive system design to meet the rapidly changing digital space, automating different operation which would otherwise require human input [14]. It might have contributed to the dynamics we observed.

Similarly, the frequency of vulnerabilities across all severity levels began to plateau after 2016 and decreased significantly after 2021. This trend suggests advancements in cybersecurity [15], driven by heightened security concerns and the implementation of stronger measures to address them. Notably, from 2021 onward, there has been a significant rise in vulnerability detection and mitigation techniques, particularly those based on machine learning and data mining for automated detection [16], [17]. The trend has manifested in maven central dependency ecosystem as well, it seems. However, the top five most frequent vulnerabilities in the ecosystem were of high severity, suggesting need for more security vigilance and mitigation measures. In addition, only five percent of the total libraries were full maintained based upon the criteria we set for determining maintenance level which is explained in method section. However, ninety percent of the libraries were only partially maintained, which requires additional effort to keep them up to date and mitigate existing vulnerabilities. This, in turn, would enhance their usability and, as a result, increase their popularity. However, distribution of libraries varies with enforcement of different criteria, which is the limitation of this approach, and the study recommend an in-depth study to determine criteria for measuring software maintenance based upon different metric that can be used to detect a library as well maintained or not.

V. CONCLUSION

In this study, we present trend and growth of various component of software ecosystem in Maven Central dependency graph. We observed a general trend of growth in frequency, speed, and vulnerabilities until 2021. Beyond 2021, the trend

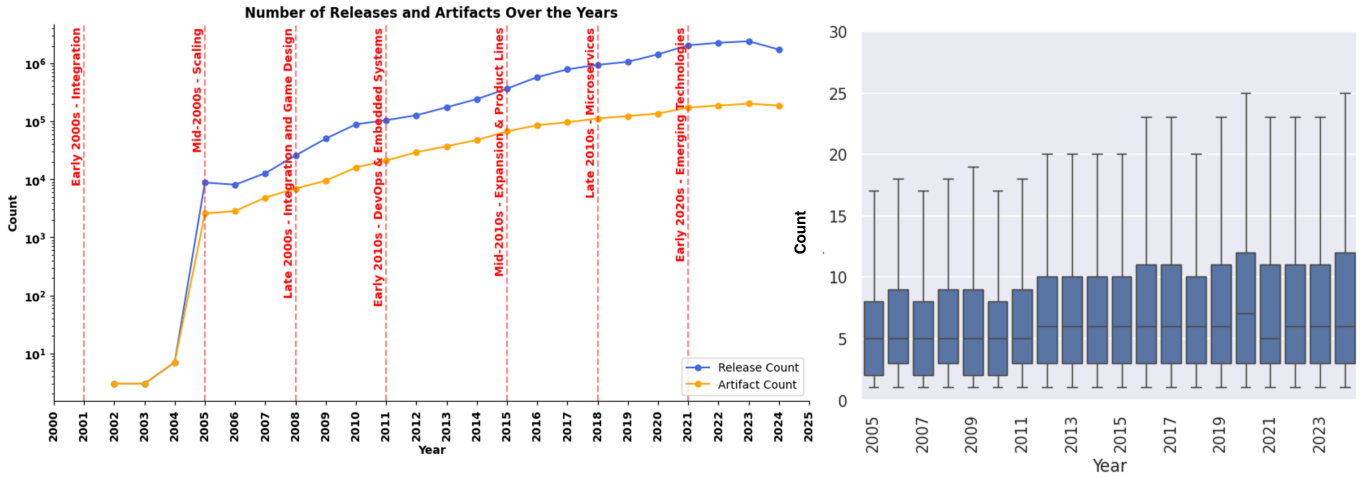

Fig. 1. **Growth of artifacts and its releases over the years (left), and box plot Distribution of dependencies of library releases over years in box plot(right)** The y-axis of the line graph represents the cumulative count, while the x-axis represents different years. The blue line and the yellow line trace the cumulative release count and artifact counts over the years, respectively. The agile software development evolution timelines are shown by the red lines, which were gathered from the work by Hoda et al. [12].Similarly, in the box plot, the y-axis represents the count of dependencies, and the x-axis represents different years. The box plot shows the upper quartile, lower quartile, and median of the distribution. Primarily, the median of the distribution was used to track the trend over the years. The plot starts from 2005 because outliers in the preceding years make it difficult to display the data in the box plot.

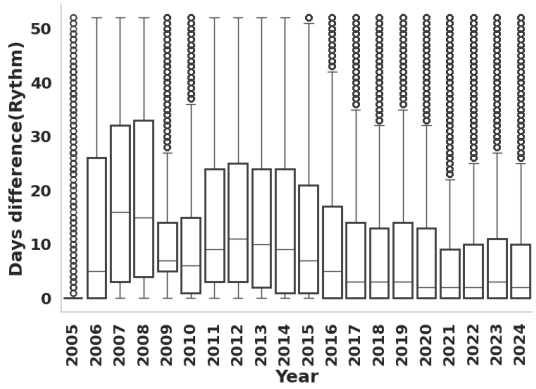

Fig. 2. **Distribution of rhythms of library release in the different years.** The X-axis represents different years, and the Y-axis represents the days' difference between consecutive releases of an artifact. The box represents the distribution of the time gap between consecutive library release of artifacts(rhythm) within a year.

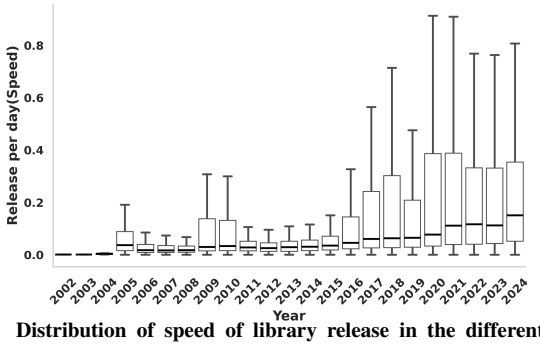

Fig. 3. **Distribution of speed of library release in the different years.** The X-axis represents different years, and the Y-axis represents the number of release per day of an artifact(seed). The box represents the distribution of the speed of a release of artifacts within a year.

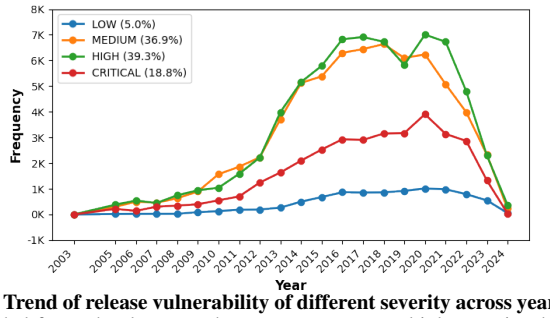

Fig. 4. **Trend of release vulnerability of different severity across years.** The CVE label for each release can have none, one, or multiple associated CVEs. The CVEs associated with each release were separated, and the severities of the individual CVEs were identified and counted. Any CVE label in an incorrect format was excluded. The line graph illustrates the trend in the count of different CVE severity levels, with each dot representing the number of cases of a specific severity type found in that particular year within the ecosystem.

began to plateau, except for the frequency of vulnerabilities of different severity, which started to decrease. This trend may be attributed to rapid advancements in the field of software development, driven by the emergence of new technologies like artificial intelligence, along with increased efforts to improve software security. The fully maintained libraries were just five percent of total libraries, suggest need for more maintained effort among the developer's community. However, the criteria used to determine the maintenance level were not well validated nor derived from standard guidelines, which limits its adoption and implementation. The study suggests the need for further in-depth research on the most prevalent vulnerabilities in the Maven Central ecosystem, as well as the development of metric-based evaluation criteria for the rapid identification of various characteristics of libraries, their releases, and associated dependencies, such as maintenance level, combined vulnerability score, etc.

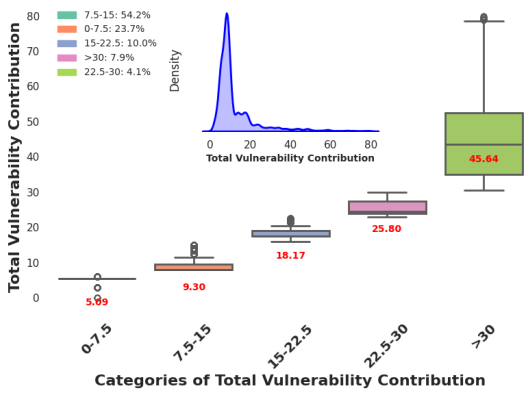

Fig. 5. **Distribution of Total Vulnerability Contribution across different contribution categories.** The total CVE contribution, or vulnerability contribution, for each release was calculated by multiplying the count of each severity type by its respective weight based on its severity level. The total vulnerability contribution represents a single severity measure, computed to reflect the overall severity impact for each release. Based on the computed values, the total contribution was categorized into five groups: 0.7–5 (low), 7.5–15 (medium), 15–22.5 (mild), 22.5–30 (moderate), and $>$ 30 (high) severity for each release. The distribution of total vulnerability contributions is shown in the box plot, along with the average, which represents the mean of all total vulnerability contributions within each category. Additionally, the percentage contribution of each vulnerability category to the total reported vulnerabilities is included, excluding releases with no vulnerabilities. The inset plot (top center) shows the density distribution of Total Vulnerability Contribution.

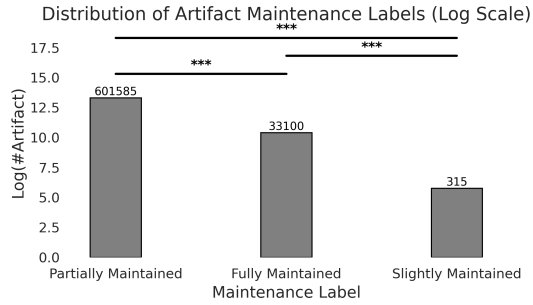

Fig. 6. **Distribution of the extent of artifact maintenance levels: fully maintained, partially maintained, and slightly maintained.** The log scale was chosen to make the bar diagram presentable. The number of artifacts subscribing to each maintenance level was shown at the top of the respective bar diagram. Kruskal-Wallis test along with Tukey's Honestly Significant Difference (HSD) at $p<0.05$ level of significance was applied to test for differences in distributions across these groups. The applied null hypothesis was that all groups have identical distributions against the alternative hypothesis that at least one group differs. Significant differences were presented as *, **, or ***, depending on the magnitude of the significance.

VI. Disclaimer Statement

The AI assistant was used to improve the grammar and sentence structure, as well as to derive summaries from the research article. However, AI was not used to generate content or write any part of the paper.

References

[1] J. Cox, E. Bouwers, M. V. Eekelen, and J. Visser, "Measuring Dependency Freshness in Software Systems," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Florence, Italy: IEEE, May 2015, pp. 109–118. [Online]. Available: http://ieeexplore.ieee.org/document/7202955/

[2] D. Jaime, J. E. Haddad, and P. Poizat, "A Preliminary Study of Rhythm and Speed in the Maven Ecosystem," 2023.

[3] S. Raemaekers, A. Van Deursen, and J. Visser, "The Maven repository dataset of metrics, changes, and dependencies," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. San Francisco, CA, USA: IEEE, May 2013, pp. 221–224. [Online]. Available: http://ieeexplore.ieee.org/document/6624031/

[4] D. Jaime, J. E. Haddad, and P. Poizat, "Navigating and Exploring Software Dependency Graphs using Goblin," 2024.

[5] C. Soto-Valero, A. Benelallam, N. Harrand, O. Barais, and B. Baudry, "The Emergence of Software Diversity in Maven Central," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. Montreal, QC, Canada: IEEE, May 2019, pp. 333–343. [Online]. Available: https://ieeexplore.ieee.org/document/8816742/

[6] A. Zerouali, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, "On the Diversity of Software Package Popularity Metrics: An Empirical Study of npm," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Hangzhou, China: IEEE, Feb. 2019, pp. 589–593. [Online]. Available: https://ieeexplore.ieee.org/document/8667997/

[7] G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "How the Apache community upgrades dependencies: an evolutionary study," *Empirical Software Engineering*, vol. 20, no. 5, pp. 1275–1317, Oct. 2015. [Online]. Available: http://link.springer.com/10.1007/s10664-014-9325-9

[8] Y. Wu, Z. Yu, M. Wen, Q. Li, D. Zou, and H. Jin, "Understanding the Threats of Upstream Vulnerabilities to Downstream Projects in the Maven Ecosystem," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. Melbourne, Australia: IEEE, May 2023, pp. 1046–1058.

[9] Jailton Junior de Sousa Coelho, "Identifying and characterizing unmaintained projects in github," Ph.D. dissertation, Universidade Federal de Minas Gerais, Aug. 2019. [Online]. Available: http://hdl.handle.net/1843/31230

[10] J. Khondhu, A. Capiluppi, and K.-J. Stol, "Is It All Lost? A Study of Inactive Open Source Projects," in *Open Source Software: Quality Verification*, E. Petrinja, G. Succi, N. El Ioini, and A. Sillitti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, vol. 404, pp. 61–79, series Title: IFIP Advances in Information and Communication Technology.

[11] M. Aggarwal, "A Study of CVSS v4.0: A CVE Scoring System," in *2023 6th International Conference on Contemporary Computing and Informatics (IC3I)*. Gautam Buddha Nagar, India: IEEE, Sep. 2023, pp. 1180–1186.

[12] R. Hoda, N. Salleh, and J. Grundy, "The Rise and Evolution of Agile Software Development," *IEEE Software*, vol. 35, no. 5, pp. 58–63, Sep. 2018.

[13] A. Decan, T. Mens, and P. Grosjean, "An empirical comparison of dependency network evolution in seven software packaging ecosystems," *Empirical Software Engineering*, vol. 24, no. 1, pp. 381–416, Feb. 2019.

[14] Mohan Harish Maturi,, Sai Sravan Meduri,, Hari Gonaygunta,, and Geeta Sandeep Nadella, "A Systematic Literature Review: The Recent Advancements in AI Emerging Technologies and Agile DevOps," *International Meridian Journal*, vol. 2, no. 2, pp. 1–23, 2020.

[15] J. Kaur and K. R. Ramkumar, "The recent trends in cyber security: A review," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 8, pp. 5766–5781, Sep. 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1319157821000203

[16] A. Ghosh, T. O'Connor, and G. McGraw, "An automated approach for identifying potential vulnerabilities in software," in *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No.98CB36186)*. Oakland, CA, USA: IEEE Comput. Soc, 1998, pp. 104–114.

[17] H. Hanif, M. H. N. Md Nasir, M. F. Ab Razak, A. Firdaus, and N. B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *Journal of Network and Computer Applications*, vol. 179, p. 103009, Apr. 2021.