A preliminary study of rhythm and speed in the maven ecosystem

Damien Jaime[1,2], Joyce El Haddad[3] and Pascal Poizat[1,4]

[1]Sorbonne Université, CNRS, LIP6, F-75005, Paris, France
[2]SAP France S.A, F-92300, Levallois-Perret, France
[3]Université Paris Dauphine-PSL, CNRS, LAMSADE, F-75016, Paris, France
[4]Université Paris Lumières, Université Paris Nanterre, F-92000, Nanterre, France

Abstract

The analysis and the evolution of dependencies are central issues in software development. They can be studied from an ego-centric perspective, *i.e.*, focusing on a given software artifact and its dependencies. But the scope can also be enlarged to take into account the whole context in which these artifacts and their dependencies evolve, *i.e.*, ecosystems. Our general objective is to study how the knowledge about these ecosystems can guide developers in evolving software artifacts and their dependencies. In this paper we perform a preliminary study of the Maven ecosystem with reference to two metrics related to software artifact releases, namely rhythm and speed.

Keywords

software evolution, software ecosystem, dependency management, releases, rhythm, speed, Maven

1. Introduction

Using dependencies to third parties is popular because it reduces development time and improves software quality [1]. Yet, this comes at a cost as these dependencies increase the architectural complexity and maintenance costs.

Raemaekers *et al.* [2] studied in 2012 third-party stability through version analysis, and concluded that depending on old third-party versions is a "maintenance debt". Updating dependencies may however lead to a dilemma, since it may cause breaking changes [3]. A trade-off has then to be found in dependency management between keeping dependencies as is, and, *e.g.*, missing security patches, or updating them, and possibly being hit by new bugs.

The study of dependencies may be achieved with a single system (ego-centric) perspective, or with an ecosystem one. The former may increase the quality of the system of interest, through the analysis of dependencies within the system (endo-dependencies) or with third parties (exo-dependencies) [4]. The latter paves the way to the empirical study of dependencies and, possibly, allows one to develop concepts and development practices based on ecosystem-wide knowledge.

BENEVOL'22: The 21st Belgium-Netherlands Software Evolution Workshop, 12-13 September 2022, Mons, Belgium

damien.jaime@lip6.fr (D. Jaime); joyce.elhaddad@lamsade.dauphine.fr (J. El Haddad); pascal.poizat@lip6.fr (P. Poizat)
https://lip6.fr/Damien.Jaime (D. Jaime); https://www.lamsade.dauphine.fr/~elhaddad (J. El Haddad); https://lip6.fr/pascalpoizat (P. Poizat)
0000-0002-7503-4606 (D. Jaime); 0000-0002-2709-2430 (J. El Haddad); 0000-0001-7979-9510 (P. Poizat)

CEUR Workshop Proceedings (CEUR-WS.org)

In 2015, Cox *et al.* proposed a metric to measure the dependency "freshness" and applied it, for a single third-party, on a whole ecosystem [5]. Thereafter, González-Barahona *et al.* proposed the "technical lag" concept, focusing on the relation to vulnerabilities and bugs [6, 7]. Since 2018, several authors have carried out studies concerning technical lag in ecosystems. This has led to analyses for Linux distributions [8, 9], Docker Hub [10], package managers [11] and more specifically for npm [12, 13, 14, 15]. In 2021, Soto-Valero *et al.* [16], presented the notion of "bloated dependencies", and analyzed their presence in the Maven ecosystem.

In this paper, alike the above-mentioned works, we take an ecosystem perspective on third-party and dependency study. Working in the context of industrial applications with both Java and Javascript dependencies, we decide to focus, first, on the Maven ecosystem, which has been less studied than the npm one. While a snapshot of Maven Central is available for study [17], dated 2018, Ochoa *et al.* have shown how knowledge and truths could change over a few years [18]. As a consequence, we propose first a study of Maven in 2022, differences with 2018 being clear in Table 1.

We want to define metrics according to the evolution in time. These metrics are part of a larger study related to the quality of dependencies in software ecosystems. The goal is to make software safer and more maintainable by improving its structural quality. This paper introduces two new metrics related to artifacts and releases in ecosystems, namely rhythm and speed. We will therefore introduce these terms first and then review some possible applications of them. We believe that these metrics will help to solve the trade-off between updating or not a dependency [19] by tracking the dynamics of a package or ecosystem over time (*e.g.* accelerating releases may indicate a need to update the dependency). The research questions that these metrics enable us to study, and the related analyses, are presented in Sections 4 and 5. Experimental data and our Python Jupyter notebooks are available at [20].

2. Metrics for the Ecosystem Analysis

A software artifact release (or release for short) in the Maven ecosystem can be identified by a triple $x : y : z$, where x is the group id, y is the artifact id, and z is the version. An artifact can indeed be denoted using only the first two parts of a release, *i.e.*, $x : y$. Given a release r, $date(r)$ denotes its release date. Given two subsequent releases $r_{i-1} = x:y:v_{i-1}$ and $r_i = x:y:v_i$ of an artifact, $time(r_i)$ denotes the number of days between the two releases, *i.e.*, $date(r_i) - date(r_{i-1})$. When r_i is the first release of an artifact, $time(r_i)$ is 0. We use $[d_1 – d_2]$ to denote the time interval between d_1 and d_2. We may use $[0 – d_2]$ to denote time up to d_2 and $[0 – \infty]$ to denote the whole time. Given an artifact a and a time interval $[d_1 - d_2]$, $releases_{[d_1 – d_2]}(a)$ denotes the time-ordered sequence $\langle r_1, \dots, r_i, \dots, r_n \rangle$ of releases of a in $[d_1 – d_2]$. Finally, given an ecosystem ϵ, $artifacts(\epsilon)$ denotes the set of artifacts in ϵ. We may now introduce our two metrics: rhythm and speed.

Definition 1 (Rhythm). *The release rhythm (or rhythm for short) of an artifact a over a time interval $[d_1 - d_2]$, denoted by $\mathcal{R}_{[d_1 – d_2]}(a)$, is given by the formula:*

$$\mathcal{R}_{[d_1 – d_2]}(a) = \langle time(r_1), \dots, time(r_i), \dots, time(r_n) \rangle \tag{1}$$

with $\langle r_1, \dots, r_i, \dots, r_n \rangle = releases_{[d_1 – d_2]}(a)$.

Definition 2 (Speed). *The average speed (or speed for short) of an artifact a over a time interval $[d_1 - d_2]$, denoted by $\overline{v}_{[d_1 – d_2]}(a)$, is given by the formula:*

$$\overline{v}_{d_1-d_2}(a) = \frac{|releases_{[d_1 – d_2]}(a)|}{(d_2 - d_1)} \tag{2}$$

where $|s|$ denotes the size of sequence s. The speed is measured in releases per day (r/d).

Speed can be lifted up to ecosystems.

Definition 3 (Ecosystem Speed). *The average speed (or speed for short) of an ecosystem ϵ over a time interval $[d_1 - d_2]$, denoted by $\overline{v}_{[d_1 – d_2]}(\epsilon)$, is given by the formula:*

$$\overline{v}_{[d_1 – d_2]}(\epsilon) = \frac{\sum_{a \in artifacts(\epsilon)} |releases_{d_1-d_2}(a)|}{(d_2 - d_1) \times |artifacts(\epsilon)|} \tag{3}$$

One can note (i) that the speed of an artifact a is the speed of an ecosystem that is composed only of a and (ii) that we also have:

$$\overline{v}_{[d_1 – d_2]}(\epsilon) = \frac{\sum_{a \in artifacts(\epsilon)} \overline{v}_{[d_1 – d_2]}(a)}{|artifacts(\epsilon)|} \tag{4}$$

3. Presentation of the Datasets

In this section, we present the two datasets we use to perform the analysis of the Maven ecosystem (see Tab. 1).

3.1. Maven Dependency Graph (MDG)

This dataset has been proposed by Benelallam *et al.* in [17]. It is a graph-based representation of a snapshot of the Maven Central repository at September 6th, 2018. Its vertices include for each release its id, the packaging type (jar, war, ear), and the release date. Two kinds of edges are used: one for the precedence relationship between releases (e.g., $x:y:1.0.1$ is next to $x:y:1.0.0$) and one for the dependency relationship between artifact releases given a context (e.g., $x_1:y_1:z_1$ depends on $x_2:y_2:z_2$ for the compiling context). As stated in [17] the data collection has been achieved by *"retrieving pom.xml files (at least one per artifact) from Maven Central and parsing them to retrieve metadata"*.

Table 1
Descriptive statistics about the MDG (2018-09-06) and the MCI (2022-06-10) datasets.

Property	MDG	MCI	Increase
Releases (unique $x:y:z$)	2,407,335	9,060,212	276%
Artifacts (unique $x:y$ in $x:y:z$)	223,478	477,707	113%
Groups (unique x in $x:y:z$)	35,699	66,139	85%
Upgrades (release evolutions)	2,183,845	8,836,734	304%
Dependency relationships	9,715,669	–	–

3.2. Maven Central Index extraction (MCI)

The MCI is available at Maven Central[1]. It contains all artifacts, updated weekly. We will use the version at June 10th, 2022. From the MCI we obtain a Lucene database using scripts available at Maven Central and then we extract CSV files from it. These CSV files are available at [20]. As for the MDG, this dataset includes release ids and release dates. However, it does not (yet) include dependency information.

It is interesting to see how the number of releases has grown in almost 4 years between the MDG snapshot and the more recent MCI one. There is an important increase, which makes it interesting to have recent data. We study the increase of ecosystem speed in Section 4.1, but it would be interesting to study this behavior in more detail, especially by comparing it with npm for example.

4. Dependency-independent Analysis

In this section, we study the Maven ecosystem independently of the dependencies between releases. We wish to address two research questions :

- **RQ1:** How do rhythm and speed evolve over time in the Maven ecosystem?
- **RQ2:** Can important events in the life of an artifact be observed through the prism of rhythm?

4.1. Analysis of the Maven ecosystem evolution

Table 1 shows an important increase in the number of releases between 2018 and 2022. This raises the question of the evolution of rhythm between these two years.

To answer this question, we took the MCI dataset and computed the distribution of times between releases ($dtbr$) respectively for all years up to 2018 and for all years up to 2022. For this we used the the following function:

$$
\begin{array}{rcl}
dtbr(i) = \{\!\{ time(r) & \mid & a \in artifacts(\epsilon) \wedge \\
 & & r \in releases_i(a) \wedge \\
 & & time(r) \neq 0 \}\!\}
\end{array}
\tag{5}
$$

where i is a date interval. This regroups all non-null times between releases for releases with a date in the interval. Here, we use the [0 – 31/12/2018] and [0 – 31/12/2022] intervals and the result is shown in Table 2.

We can consider that, regarding the date of the corresponding release, a release time around Q1 is fast, around Q3 is slow, and around the median is medium compared to the ecosystem. We can see that, within 4 years, the time between releases decreased which suggests an overall acceleration.

Let us now study the general evolution of both rhythm and speed for all years between 2005 and 2022. We followed the same approach as before for aggregating the rhythm, this time

[1] https://maven.apache.org/repository/central-index.html

Table 2
Comparison of time between releases in days up to 2018 and up to 2022.

	MCI up to 2018	MCI up to 2022
Medians	14.0	9.0
Means	43.9	35.6
Q1	4.0	2.0
Q3	43.0	32.0
Minimums	1	1
Maximums	101	77
Outlier maximums	4341	5206

focusing only on years of interest (instead of up-to), so given some year y we use Function (5) with i = [01/01/y–31/12/y] (instead of [0–31/12/y]).

The result is given in Figure 1. Each box-plot corresponds to the aggregated rhythm for the year of interest, while speed is computed using Formula 2 with yearly intervals. Note that 2022 is taken into account only until June, 10^{th}.

We can observe an accelerating trend except for the period between 2009 and 2012 when there is a deceleration. We can also distinguish different parts. There is an acceleration between 2005 and 2009, a deceleration between 2009 and 2012, a new acceleration between 2012 and 2019, and finally a more important acceleration since 2019. It would be interesting to study the possible causes of these variations (new Java or Maven release? COVID-19 emergence?). While this preliminary study does not aim at finding causality relationships, these questions could be addressed in a later study.

If we observe now the evolution of rhythm at the artifact level, we get the result presented in Figure 2a. In this figure, we have computed the average time to release the x^{th} version of

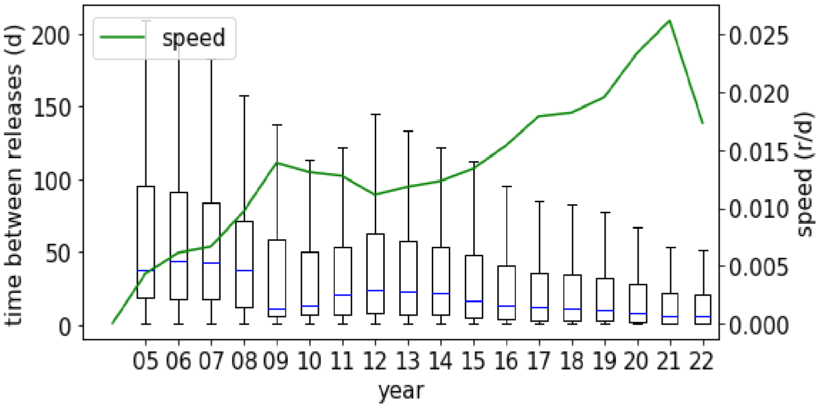
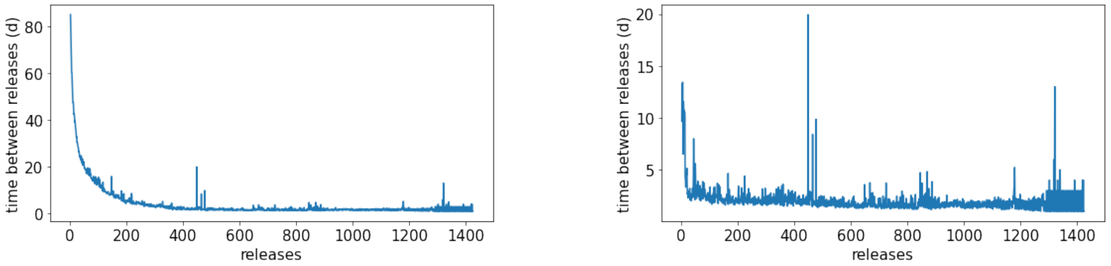


Figure 1: Evolution of aggregated rhythm and speed in the Maven ecosystem between 2005 and 2022.



(a) all artifacts

(b) only artifacts with more than 400 releases

Figure 2: Average time to release i^{th} versions.

artifacts, *i.e.*, using the following formula:

$$f_6(x) = \frac{\sum_{r\in x^{ths}(\epsilon,x)} time(r)}{|x^{ths}(\epsilon,x)|} \tag{6}$$

where x is the rank of the version (*e.g.*, 4 means it is the fourth) and $x^{ths}(\epsilon, x) = \bigcup_{a\in artifacts(\epsilon)} releases_{[0-\infty]}(a)[x]$.

From the figure we can see, *e.g.*, that on average the second release of an artifact takes about 85 days to release, while its 50th release takes only 20 days.

It seems that the more mature an artifact is, the more its rhythm tends to accelerate. But the reality is more complex than that. It is important to distinguish several groups in the ecosystem and to compare them independently. Indeed the artifacts with many releases have a different rhythm than those with few releases, and the proportion of artifacts with 10 releases is not at all the same as those with 400 releases.

Figure 2b, focuses only on the (931) artifacts with more than 400 releases. We can see that they have immediately a very low time between releases. It would be interesting to define groups of artifacts in the ecosystem to be able to associate a given artifact to the right group and compare it (based on rhythm and speed) more precisely.

4.2. Analysis of the evolution of an artifact: log4j-core

We will now see if rhythm allows us to observe changes in an artifact's life. Each artifact has a different rhythm, so to know if a release was faster than the others we have to perform an analysis at the artifact level.

Take the example of Apache Log4j2 (more precisely, artifact org.apache.logging.log4j:log4j-core), the most used logger library in the Java world. On December 9th, 2021, a level 10 security fault (CVE-2021-44228) was discovered which impacted a large number of artifacts depending on it. The rhythm distribution of Log4j2 before the issue discovery (last version on March 7th, 2021) was medians: 63.0, Q1: 26.5, Q3: 124.0, which position it as a slow library compared to the Maven ecosystem (see Sect. 4.1). At the time of writing this paper, the last version is dated February 23rd, 2022, and the rhythm distribution is medians: 48.5, Q1: 19.0, Q3: 114.2. We can see that the security issue has had an impact.

The blue plot in Figure 3 shows the rhythm of the project since 2020. If we focus on [07/03/2021 − 23/02/2022], we get [277, 4, 1, 4, 2, 1, 6, 1, 1, 56]. We observe a sharp drop on

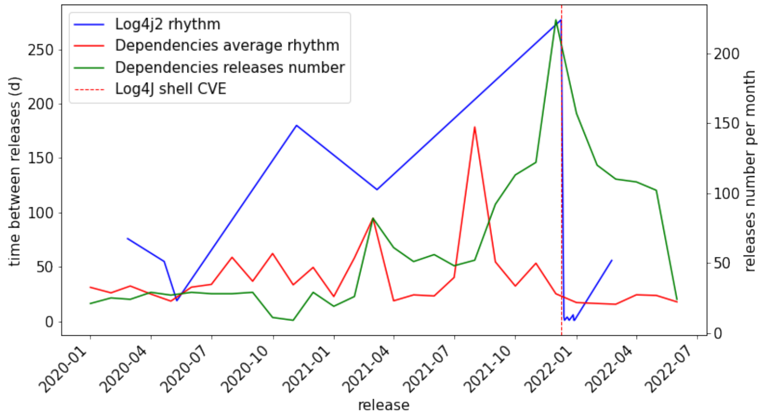


Figure 3: Evolution of the Log4j2 rhythm and relation to its dependents.

the same day as the issue was discovered. Speed had already suffered decelerations in the past, but this one is distinguished by the fact that the drop is vertical and is followed by a plateau. We can therefore say that there is a strong reactivity, a period of stabilization and then a beginning of a return to normal.

Rhythm allowed us to observe a change in the life of Log4j2, through the imprint it had. Yet, it would be interesting to analyze the whole Maven ecosystem to search for such patterns and see, *e.g.*, if there is a correlation between CVEs and rhythm changes.

5. Dependency-aware analysis

In this section, we bring dependencies into the big picture. We wish to address one research question:

- **RQ3:** Does the rhythm of an artifact have an impact on the rhythm of artifacts that depend on it?

Since we have seen in Section 4.2 how the rhythm of Log4j2 evolved concerning CVE-2021-44228, we will address this question on this artifact.

To recover dependency information, we searched, in the Maven Repository[2], for the uses of the last version of log4j-core before the discovery of the issue, namely version 2.14.1 released on March 12^{th}, 2021. Not having found either an API or export mechanisms for this, we created a script for browsing and parsing the Web pages to find the dependent artifacts. This gave us 500 different releases, for 149 distinct artifacts. The data is, again, available at [20].

In Figure 3, we have seen that the blue plot is the Log4j2 rhythm from 2020 to 2022. The red plot shows the average rhythm of the Log4j2 dependents on the same period. Finally, the

[2] https://mvnrepository.com/

green plot shows the total number of releases of these dependents. All three plots are computed monthly.

We notice an increase in the number of releases of dependents in the month of the CVE-2021-44228 discovery. In December 2021, 224 artifacts have made a release, the release time median is 5 days, with a minimum value at 1 day and a maximum value at 594 days. Yet, the average rhythm of these dependents does not show a significant variation. Possibly, artifacts that had not been released for a long time and those that had just been released compensate for each other. Thus, we can conclude that the average rhythm is not a tool precise enough to study the effect of an artifact on its dependents.

6. Limitations & Threats to Validity

First of all, our analyses are based on extractions from Maven Central. Results may therefore not apply to all of the Maven or Java ecosystem. Broadly, we need to study if our concepts are amenable to software ecosystems in general.

We made artifact-centered experiments on Log4j2, with time intervals centered around the CVE-2021-44228 discovery. This required to use the MCI dataset and own-made dependency extraction. To generalize these experiments, we would have either to use the MDG dataset and limit to 2018, update MDG, or add dependencies in our MCI extraction.

In experiments based on the time between release values or rhythm, we do not take into account releases with a time value of 0 (any r such that $time(r) = 0$). This occurs for the first releases of artifacts or if an artifact has several releases on the same day. We should study the impact of this choice.

Finally, a need for categorization arises when comparing a specific artifact to the ecosystem. This is due to possible heterogeneity between artifacts (we have seen an example of this in Fig. 2a and 2b). It would be interesting to use labeling information (*e.g.*, the Maven categories of artifacts) or to use clustering to be able to relate a given artifact to close ones.

7. Conclusion & Perspectives

In this paper, we have studied the rhythm and speed of Maven Central until June 10^{th}, 2022. We have explored possible applications of these metrics. It will be necessary to deepen their applications, but we believe that they can be useful in the management and analysis of dependencies. We have observed a general trend toward a decrease in time between releases, and accordingly an increase in speed, even as the number of artifacts in the ecosystem increases. We were also able to observe a correlation between the number of releases of an artifact and its rhythm. Finally, we have seen that an event in the life of an artifact could have a visible imprint on its rhythm and that we could observe a change of behavior in the artifacts depending on it, even if not on their rhythm.

This preliminary study paves the way to further analyses such as clustering based on rhythm and speed. We have first results showing a possible relation between the number of dependents of an artifact and its speed. We have not been able to talk about it in this paper, due to room running short, but our experimentation on the MDG dataset is available at [20].

A first perspective is to study how the knowledge of the rhythm and speed of one's system dependencies, including relative to the ecosystem, may help in managing these dependencies, *e.g.*, prioritizing upgrades or selecting alternatives. Another perspective is related to the comparison of the Maven and npm ecosystems, first from the rhythm and speed perspective. We also want to extend the MDG dataset dependency graph model (or define our own) in three directions. First, we want to support metrics related to aging dependencies, such as freshness [5], and study how they compose transitively (along dependencies of dependencies). The second extension is concerned with the expressiveness of dependency requirements. Maven for example supports version ranges. Multiple dependency versions are also possible up to transitivity and Maven has its own resolution algorithm for this. A consequence is that the effective dependency used at some date d_1 may be different from the one used at d_2, even if the POM file has not changed in between. A dependency graph should support expressive dependency constraints rather than hardcoded dependency relations as in the MDG dataset, and realization of constraints (fixing the version to be used) only when needed. The last extension is to include other parts of the ecosystem in its dependency graph. Examples include the compiler, virtual machine, and build system versions, which have an impact, *e.g.*, on the success of a system build.

Acknowledgments

This work is funded by an ANRT PhD grant, number 2021/0047.

References

[1] R. Abdalkareem, O. Nourry, S. Wehaibi, S. Mujahid, E. Shihab, Why do developers use trivial packages? An empirical case study on npm, in: Proc. of the Joint Meeting on Foundations of Software Engineering, 2017, pp. 385–395.

[2] S. Raemaekers, A. van Deursen, J. Visser, Measuring software library stability through historical version analysis, in: Proc. of the International Conference on Software Maintenance, 2012, pp. 378–387.

[3] S. Raemaekers, A. van Deursen, J. Visser, Semantic versioning and impact of breaking changes in the maven repository, Journal of Systems and Software 129 (2017) 140–158.

[4] V. Musco, M. Monperrus, P. Preux, A generative model of software dependency graphs to better understand software evolution, arXiv preprint (2014). doi:10.48550/ARXIV.1410.79211.

[5] J. Cox, E. Bouwers, M. van Eekelen, J. Visser, Measuring dependency freshness in software systems, in: Proc. of the International Conference on Software Engineering, 2015, pp. 109–118.

[6] J. M. González-Barahona, P. Sherwood, G. Robles, D. Izquierdo, Technical lag in software compilations: Measuring how outdated a software deployment is, in: Proc. of the International Conference on Open Source Systems: Towards Robust Practices, 2017, pp. 182–192.

[7] J. M. González-Barahona, Characterizing outdateness with technical lag: An exploratory

study, in: Proc. of the International Conference on Software Engineering, Workshops, 2020, pp. 735–741.

[8] D. Legay, A. Decan, T. Mens, On package freshness in linux distributions, 2020. doi:10.48550/ARXIV.2007.16123.

[9] D. Legay, A. Decan, T. Mens, A quantitative assessment of package freshness in linux distributions, 2021. doi:10.48550/ARXIV.2103.09066.

[10] A. Zerouali, T. Mens, A. Decan, J. M. González-Barahona, G. Robles, A multi-dimensional analysis of technical lag in debian-based docker images, Empirical Software Engineering 26 (2021) 1–45.

[11] J. Stringer, A. Tahir, K. Blincoe, J. Dietrich, Technical lag of dependencies in major package managers, in: Proc. of the Asia-Pacific Software Engineering Conference, 2020, pp. 228–237.

[12] A. Decan, T. Mens, E. Constantinou, On the evolution of technical lag in the npm package dependency network, in: Proc. of the International Conference on Software Maintenance and Evolution, 2018, pp. 404–414.

[13] A. Zerouali, E. Constantinou, T. Mens, G. Robles, J. M. González-Barahona, An empirical analysis of technical lag in npm package dependencies, in: Proc. of the International Conference on Software Reuse, 2018, pp. 95–110.

[14] A. Zerouali, T. Mens, J. M. González-Barahona, A. Decan, E. Constantinou, G. Robles, A formal framework for measuring technical lag in component repositories - and its application to npm, Journal of Software: Evolution and Process 31 (2019) e2157.

[15] F. R. Cogo, G. A. Oliva, A. E. Hassan, An empirical study of dependency downgrades in the npm ecosystem, IEEE Transactions on Software Engineering 47 (2021) 2457–2470.

[16] C. Soto-Valero, N. Harrand, M. Monperrus, B. Baudry, A comprehensive study of bloated dependencies in the maven ecosystem, Empirical Software Engineering 26 (2021) 1–44.

[17] A. Benelallam, N. Harrand, C. Soto-Valero, B. Baudry, O. Barais, The maven dependency graph: A temporal graph-based representation of maven central, in: Proc. of the International Conference on Mining Software Repositories, 2019, pp. 344–348.

[18] L. Ochoa, T. Degueule, J.-R. Falleri, J. Vinju, Breaking bad? Semantic versioning and impact of breaking changes in maven central: An external and differentiated replication study, Empirical Software Engineering 27 (2022) 1–42.

[19] D. Meil, Log4j and the thankless high-risk task of managing software component upgrades, 2022. URL: https://cacm.acm.org/blogs/blog-cacm/257897-log4j-and-the-thankless-high-risk-task-of-managing-software-component-upgrades/fulltext.

[20] D. Jaime, J. El Haddad, P. Poizat, Data and notebooks for the "A Preliminary Study of Rhythm and Speed in the Maven Ecosystem" paper., 2022. URL: https://github.com/DaJaime/BENEVOL2022.