EMSE-D-20-00026R1: Revision

Dear Associate Editor, *Springer Empirical Software Engineering*,

We would like to submit a second revision for our manuscript entitled **"A comprehensive Study of Bloated Dependencies in the Maven Ecosystem"**. We thank the reviewers for their detailed recommendations. We have addressed all their comments, which has improved the quality of our manuscript.

In this revision:

- We have computed and compared the ratio of bloat for each type of dependency relationship (direct, transitive, and inherited); the results are discussed in Section 5.1 and shown in new Figure 11 in the manuscript.
- We have clarified our data selection process with more details and new Figure 5 in the manuscript.
- We have performed statistical tests indicating a significant positive correlation between the number of direct dependencies and transitive dependencies. These results are discussed in the manuscript in Section 5.2.1.
- We have performed experiments that show that our dataset is representative, compared to 9 random samples of the Maven Dependency Graph (MDG). These results are discussed in Comment 3 of the revision.
- We have performed a linear regression model that estimates the percentage of bloated dependencies in an artifact based on the height of its dependency tree. The results are discussed in Comment 7 of the revision.

We have corrected all the minor remarks and typos that the reviewers brought to our attention. In the following pages, we give detailed answers to each of the comments of **Reviewer #3**. Quotes from the review are included in boxes, our answers follow the boxes. All changes are highlighted in blue in the revised version of the paper (except typos). We do not discuss the remarks of reviewers #1 and #2, who are satisfied with the changes that we made as part of the previous major revision.

In case of requiring any further information, please do not hesitate to contact us.

Sincerely yours,

César Soto-Valero

On behalf of Nicolas Harrand, Martin Monperrus, and Benoit Baudry

Response to Reviewer #3

> **Comment 1** "The authors clarified why they selected 14,699. However, why are direct dependencies a good measure for the other types of dependencies that they also investigate in RQ1 and RQ2, e.g. for transitive dependencies?"

Direct dependencies are explicitly declared in the POM of Maven projects. They are added, maintained, and controlled by the developers of the artifact. On the other hand, transitive dependencies are maintained by other teams, and then resolved by Maven. As a result, developers are not aware of the transitive dependencies that they pull from external repositories when declaring a direct dependency [1]. Therefore, direct dependencies are representative of the initial intentions of the developers with respect to code reuse, and we consider them as an appropriate metric to study bloat.

We have provided more details to support our choice of using direct dependencies as an indicator of reuse intention in Section 4.2.1 of the manuscript (page 13).

> **Comment 2** "Furthermore, they state that they applied additional selection criteria on the 14,699 sampled artifacts which resulted in a data set of 9,770 artifacts that they study. Are the 9,770 still representative? This unfortunately remains unclear (also in the plot of the comment) and I'd ask the authors to clarify this or apply the selection criteria earlier in the filter process. The authors may even include the corrected plot in the paper."

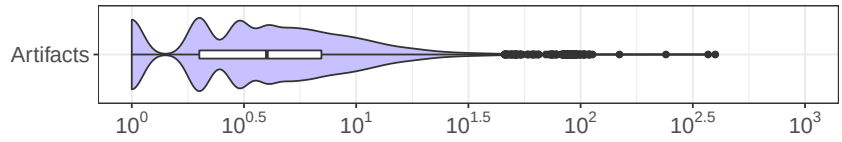
We thank the reviewer for this remark. Figure 1 shows the distribution of the number of direct dependencies at each step of our data selection process. Figure 1a shows the distribution of the 2,407,335 artifacts (1st Qu. = 2, Median = 4, 3rd Qu. = 7) in the MDG. Figure 1b shows the distribution of the sample of 14,699 artifacts (1st Qu. = 2, Median = 3, 3rd Qu. = 7) obtained from Figure 1a. Figure 1c shows the distribution of artifacts (1st Qu. = 3, Median = 5, 3rd Qu. = 9) resulting from applying additional filtering criteria to the set of artifacts in Figure 1b.

As we observe, the density of artifacts with a number of direct dependencies in the range [3, 9] in our dataset (Figure 1a) is higher than in the MDG (Figure 1c). This is a direct consequence of the selection criteria that we applied to ensure artifacts complex enough (in terms of their number of dependencies) as we mention in the manuscript: "*The subjects have at least one direct dependency with compile scope*". This filter removes artifacts that contain only dependencies that are not shipped in the JAR of the artifact (e.g., *test* dependencies). Therefore, the 9,770 artifacts used as study subjects are representative of the artifacts in Maven Central that include third-party dependencies in the JAR.

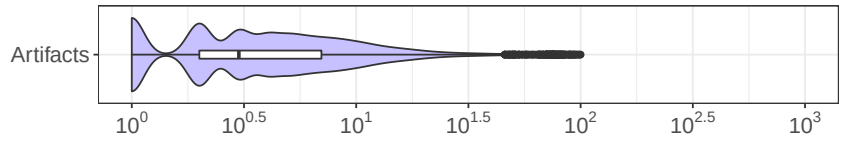
We have included Figure 1 in the Section 4.2.1 of the manuscript (page 14) to clarify our data selection procedure.

> **Comment 3** "Furthermore, while the authors state in their comments that there is a positive correlation between the number of direct dependencies and the number of transitive dependencies, it is still not clear why this criterion is a good measure for representativeness of the whole Maven universe. The authors may at least select a random sample (while still filtering with their additional criteria to avoid including obviously irrelevant artifacts) and investigating this sample in comparison with their current sample."

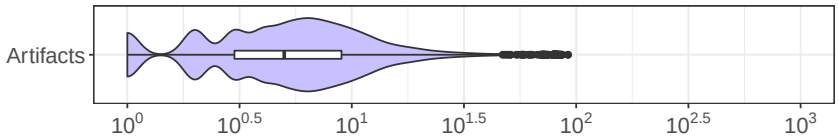
We assume that the reviewer refers to the representativeness of the dataset of 9,770 artifacts, selected based on their number of direct dependencies, with respect to a random set of artifacts sampled from the MDG.



(a) Distribution of the number of direct dependencies in the original MDG dataset of 2,407,335 artifacts.



(b) Distribution of number of direct dependencies in the sample of 14,699 artifacts obtained from (a).



(c) Distribution of the number of direct dependencies in the filtered sample of 9,770 artifacts obtained from (b).

Figure 1: Distribution of the number of direct dependencies of the artifacts at the different stages of the data filtering process.

To verify if the artifacts in our dataset differ from the MDG with respect to their distribution of direct dependencies, we have randomly sampled (without replacement) 9 sets of artifacts with the same cardinality as our dataset. As indicated by the reviewer, the sampled artifacts were filtered using our additional criteria to avoid including irrelevant artifacts.

Figure 2 shows the results obtained. As we observe, the distributions are very similar. Therefore, we can conclude that our sample is representative of the whole MDG (with respect to the total number of direct dependencies per artifact).

Comment 4 "As a response to comment 3.14 the authors state that their approach for finding usage can be seen as "a form of slicing". As a consequence, please either cite an already existing approach that was implemented or highlight the differences to those existing approaches. This will help the reader to better understand the novelty of the used approach."

We thank the reviewer for this remark. We have added references related to program slicing techniques in Section 7.1 of the manuscript (page 42).

Comment 5 "Concerning comment 3.15, the authors added a mapping which helps the reader to understand how the methodology maps to the results. However, the why and how of measuring those metrics, especially the complexity, is still missing. For example, as I mentioned in the initial review, why is the height of a dependency tree considered a good measure for complexity? If the project declares the same dependencies in the project itself, the same problems can occur and I would argue the complexity is equal to the other situation. This brings me to the conclusion that this measure might not be a good indicator for complexity."

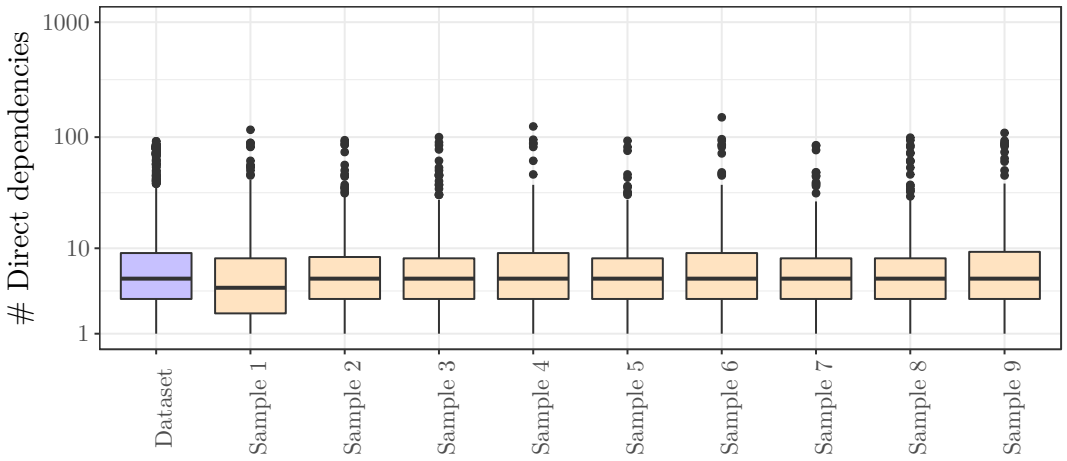
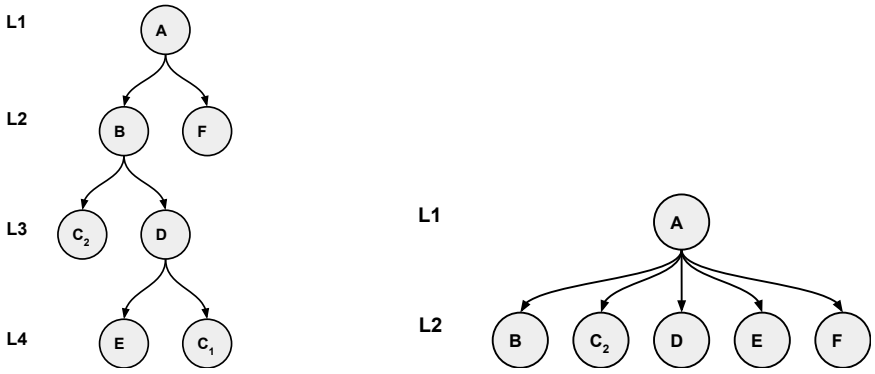


Figure 2: Distributions of the number of direct dependencies in the artifacts used in our study, and 9 random samples from the MDG dataset of 2,407,335 artifacts.



(a) Example of dependency tree with four levels (height = 3).

(b) The dependency tree of Figure 3a with two levels (height = 1).

Figure 3: Example of a dependency tree with four levels of dependencies (a), and two levels of dependencies (b).

In RQ2, we use two metrics to estimate the complexity of the dependency tree: the number of transitive dependencies, and the height of the tree. Our goal is to determine to what extent these two types of complexity measures are related to bloated dependencies.

The number of transitive dependencies in the tree indicates how many dependencies are not handled directly by developers. Transitive dependencies are not declared in the POM, which means that developers rely on the Maven dependency resolution mechanism for managing them. The height of the dependency tree, on the other hand, is a measure of the level of abstraction of this type of complexity in the tree. The intuition is that the transitive dependencies that are deeper in the dependency tree are more likely to be bloated since developers have less control of them. We have added this motivation in Section 4.2.1 (page 15).

To illustrate why we use the height of the dependency tree as a measure of complexity, we consider the example of dependency tree presented in Figure 3, which shows two different reuse architectures for a Maven project **A**. Figure 3a shows the dependency tree of an artifact **A** with four levels of dependencies (**L1**, **L2**, **L3**, and **L4**), and a height = 3. The project **A** in this tree has two direct dependencies (**B** and **F**) and four transitive dependencies ($\mathbf{C}_2$, **D**, **E**, and $\mathbf{C}_1$). Dependencies $\mathbf{C}_1$ and $\mathbf{C}_2$ represent two distinct versions of dependency **C**. On the other hand, Figure 3b shows

the dependency tree of **A**, but with the difference that it has no transitive dependencies, i.e., all the dependencies are located in the second the level of the tree.

From the perspective of developers, the dependency tree of Figure 3a is more complex and difficult to maintain than the tree in Figure 3b. In the tree of Figure 3a, developers have no control of the version of the transitive dependencies used, since all these transitive dependencies depend on the version of **B** declared in the POM (not the project itself). For example, if developers do not use, either directly or indirectly, the functionalities of $\mathbf{C}_2$, they still have to carry all the classes of this dependency in the classpath of the project. The dependency **D** uses $\mathbf{C}_1$, which is a different version of the dependency used by **B**. This situation may cause a dependency conflict (a.k.a., *dependency hell*). Furthermore, the POM of project **A** only declares two dependencies (**B** and **G**), hence, the rest of transitive dependencies are hidden from the sight of developers. In the case of large dependency trees, with thousands of transitive dependencies, this constitutes a complexity burden that is difficult to handle by humans. Indeed, our results reveal one of the problems associated to the existence of such a high number of transitive dependencies: the emergence of bloat.

On the other hand, Figure 3b shows the same project **A** of Figure 3a, but in this case developers declare all the dependencies directly in the POM. In this case, developers have full control over the version of the dependencies used in the project. For example, if they wanted to use version 1 of dependency **C**, all they need to do is to declare $\mathbf{C}_1$ instead of $\mathbf{C}_2$ in the POM. Furthermore, the dependencies in Figure 3b are more easy to locate and maintain since they are explicitly visible in the POM, which also helps new contributors to understand the codebase of the project as well.

> **Comment 6** "Based on this, the answers to RQ1 and RQ2 are still not really surprising but expected. Although in general, of course, there is no problem with unsurprising results, in this case, the effects seem to be due to the way how the experiments/measures/questions are designed. For example, referring to RQ1: There are usually a lot more transitive dependencies than direct dependencies, obviously. The result that there are also more bloated transitive dependencies seems rather shallow as this might be the effect of the higher number of transitive dependencies. If the author measured this as a relative value please make this clear in the text. Otherwise, please consider to change the measurement to make the results more expressive."

As suggested by the reviewer, we compute the ratio of the status of the dependency relationships (used or bloated) for the three types of dependency relationships studied (direct, transitive, and inherited). Figure 4 shows the results obtained. As we observe, the ratio of bloated-transitive dependencies is still the highest (82.75%), followed by bloated-inherited dependencies (61.79%). As noticed by the reviewer, the percentage of bloated-direct dependencies (34.23%) is more significant if we consider this number with respect to the total number of direct dependencies.

We have added Figure 4 and the corresponding text to Section 5.1 of the manuscript (page 21), as part of our answer to RQ1.

> **Comment 7** "Similar to this, the author state in the answer to RQ2 "the higher a dependency tree, the more bloated-transitive dependencies;". The question that I raise here is whether this is really a cause-effect relation and whether the height itself is the factor causing more bloated transitive dependencies. This is not shown in my opinion.

We thank the reviewer for raising this discussion. According to the Spearman's rank correlation test, there is a significant positive correlation between the height of the dependency tree and the percentage of bloated dependencies in the studied artifacts ($\rho = 0.54$, p-value < 0.01). We have added this result to Section 5.2.1 of the manuscript (page 27).

We performed a linear regression analysis to estimate the percentage of bloated dependencies of an artifact based on the height of the dependency tree. Table 1 shows the coefficients of the predictor variable in the model ($R^2 = 0.3$, RSE $= 22.9$). Column 2 shows the estimate of regression

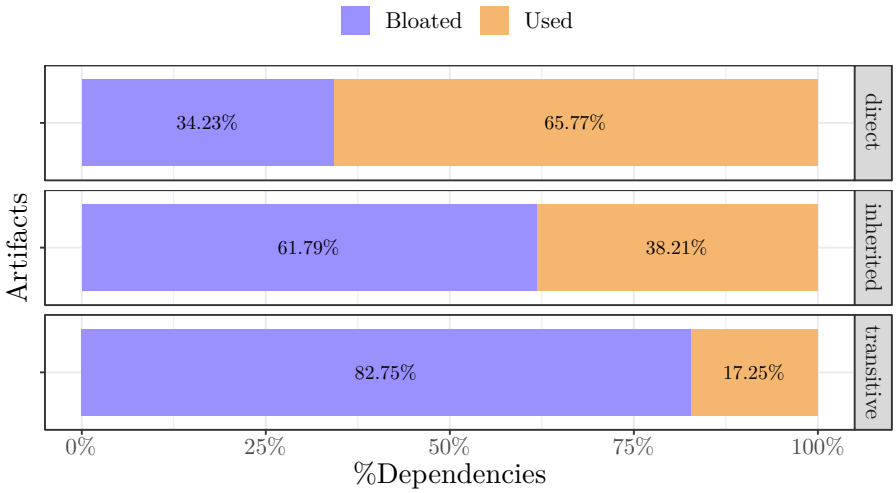


Figure 4: Ratio per usage status of the 723,444 dependency relationships analyzed.

Table 1: Table of linear coefficients to predict the percentage of bloated dependencies in an artifact based on the height of the dependency tree..

	Estimate	Std. Error	t-value	p-value
(Intercept)	26.2268	0.5670	46.25	2e-16
height	8.8585	0.1379	64.25	2e-16

beta coefficients, column 3 shows the standard error, columns 4 and 5 show the associated t-statistic and p-values. According to the p-values, changes in the variable *height* of the tree is significantly associated to changes in the percentage of bloated dependencies (p-value < 0.01). Based on this analysis, the equation of the model can be written as: $\%Bloat = 8.86 * height + 26.22$.

As we mention in the manuscript, the height of the tree is not the only factor that causes the bloat (as is evidenced by the values of R^2 and RSE obtained in the model). For example, in Section 5.2.1 we determine that the number of transitive dependencies is another essential factor. However, we found a positive relationship between both variables and studied the phenomenon based on the empirical evidence obtained.

Comment 8 "Furthermore, the authors state that "bloat is more pervasive in multi-module than single-module artifacts". Again, is the effect because of the multi-module nature of the project or simply because multi module often implies a larger project which often implies more dependencies? I'm not fully convinced that the implications are fully backed by the shown results.

In the first paragraph of Section 5.2.2, we mention that, in general, the difference of bloat between multi-module and single-module is small ("multi-module artifacts have slightly more bloat than single-module, precisely 3.1% more"). However, we observed a "shift" of bloated-direct and bloated-transitive dependencies into bloated-inherited dependencies (Figure 16 in the manuscript), and explained the possible causes and consequences according to our experience of the Maven dependency management mechanisms and the practice of software development (Section 5.2.2).

We have clarified the conclusion derived from this result in the summary box presented at the end of Section 5.2.2 of the manuscript (page 30).

Comment 9 "In the response to comment 3.20, the authors point to Table 4 which shows the number of bloated transitive dependencies that are removed when a bloated direct dependency is removed. I'd like to rephrase my concern about the numbers presented before (e.g. 75.1%) to clarify: Does this number respect the fact that a lot of transitively declared bloated dependencies get removed if a directly declared bloated dependency gets removed? If not, the authors may want to add this value to the results."

RQ1 is an empirical analysis of 9,639 DUTs in which we quantify bloat and categorize it with respect to the type of dependency (direct, inherited, transitive). In this research question, we do not make any assumption about the actions that developers would take in order to handle bloated dependencies. Consequently, we do not consider that bloated-direct dependencies have been removed before counting the ratio of bloated-transitive. Meanwhile, RQ3 focuses on how developers handle bloated-direct dependencies. Table 4, reports on the number of bloated-direct dependencies that have been removed by the developers after we informed them about the presence of this bloat. In the table, we also indicate the number of transitive dependencies that have been removed as a consequence of this action from the developer.

Comment 10 The authors may think of swapping definition 3 and 4 as definition 3 makes use of the DUT which is introduced in definition 4."

We have arranged definitions 3 and 4 according to this suggestion.

References

[1] Russ Cox. Surviving software dependencies. *Communications of the ACM*, 62(9):36–43, 2019.