Thesis Data Analysis

João Lourenço

Contents

The research goal of this thesis, as stated in the introduction, is the following:

How do monolith microservices identification approaches that use the monolith development history based representations perform when compared with approaches that use the monolith functionalities sequences of accesses representation?

To address this research question, we generate a large set of decompositions and analyse how the decomposition qualities vary depending on different monolith representations and similarity measures.

Whenever we create a decomposition, we choose the weights to attribute to our similarity measures. If we choose 0 as the weight for the commit and the author similarity, then the decomposition is created using only data from the sequences of accesses representation. On the other hand, if we choose 0 as the weight for all four sequences of accesses measures (access, read, write, sequence), then the decomposition is created using only data from the development history based representations. In the remaining combinations, the decompositions are created with data from both sources. By filtering the results of all generated decompositions, we can obtain five distinct groups of decompositions based on the weights used to create them: only data from the file changes representation; only data from the changes authorship representation; only data from the sequences of accesses representation; only data from the file changes representation and changes authorship; data from all representations. This allows us to then compare the groups' quality metrics and draw conclusions.

The comparisons will first be made by evaluating the median and the dispersion of each quality metric in each of these five groups, which gives us an overview of how, on average, each representation behaves. We also evaluate the median values of the metrics in the case of the best decompositions - that is, the decomposition of each codebase with the best value for each metric. This gives us a different perspective by focusing on which group performs best when the ideal weights for the similarity measures are found. Finally, we assess if

our findings hold when we compare codebases with more commits and authors than the mean with codebases with less commits and authors than the mean. This is done by considering the best decompositions, as it gives greater strength to our findings.

The comparisons are, initially, done visually through the analysis of boxplots. Whenever it's not visually obvious that there is a large difference between groups, and considering that there are different amounts of decompositions in each group and the quality metric values don't follow a normal distribution, we use a Welch T-test with the following hypotheses:

- H_0 - There are no significant differences between the mean ${QUALITY METRIC} of ${GROUP} decompositions and ${OTHER GROUP}.
- H_1 - The mean ${QUALITY METRIC} of ${GROUP} decompositions is greater than ${OTHER GROUP}.

1 Codebase selection and characterization

The sequences of accesses monolith representation we are comparing was developed in [1]. In that work, a total of 121 codebases using the Hibernate ORM were selected, according to the following procedure:

1. Get all GitHub repositories that list the Spring Data JPA library as a dependency;
2. Filter out repositories that did not contain at least 5 files whose name ended in `Controller.java`, and at least 5 files whose name did not contain `Dao` or `Repository`.
3. The remaining repositories were ordered by the number of GitHub stars, and 118 codebases were manually selected. Repositories from lessons or tutorials, and repositories that did not use just Spring Data JPA were disregarded.

The authors from [1] made available a .zip file with the source code of all codebases, as well as the collected sequences of accesses. This data was gathered in 2020, and most of the codebases have maintained an active development cycle since then. As such, the collected data might not be entirely accurate to the present day version of the codebase, so we cannot just clone the codebase at the latest commit and collect data related to the development history. Therefore, for the sake of our comparisons being as accurate as possible, we filter the codebases made available according to the following criteria:

1. We can only know the latest commit of each codebase if a `.git` subdirectory is available. Therefore, we filter out all folders that do not contain this directory.
2. As we are looking for an active development history, we filter out all codebases with less than 100 commits and less than 2 authors. The shell command `git rev-list --count HEAD` lists the number of commits reachable from the `HEAD` commit, and corresponds to the number of commits displayed in the GitHub page of the repository. The number of authors is obtained with the command `git log --pretty='%ae' | sort | uniq | wc -l`, which lists all author's emails from all commits, sorts them, removes duplicated ones, and then returns just the number of authors.
3. We manually discarded a few codebases that did not follow assumed conventions, like the file of an entity having the same name as the entity. For example, a `BookDepository` class in a `Bookdepository.java`.

After this filtering, we ended up keeping a total of 28 codebases. Figure 1 displays the number of functionalities as a function of domain entities, as well as the number of authors as a function of commits, and the number of entities as a function of commits. Although the number of functionalities tends to increase as more entities exist, there isn't such a clear relationship between the number of authors and the commits of a codebase, as well as the number of commits and the number of entities.

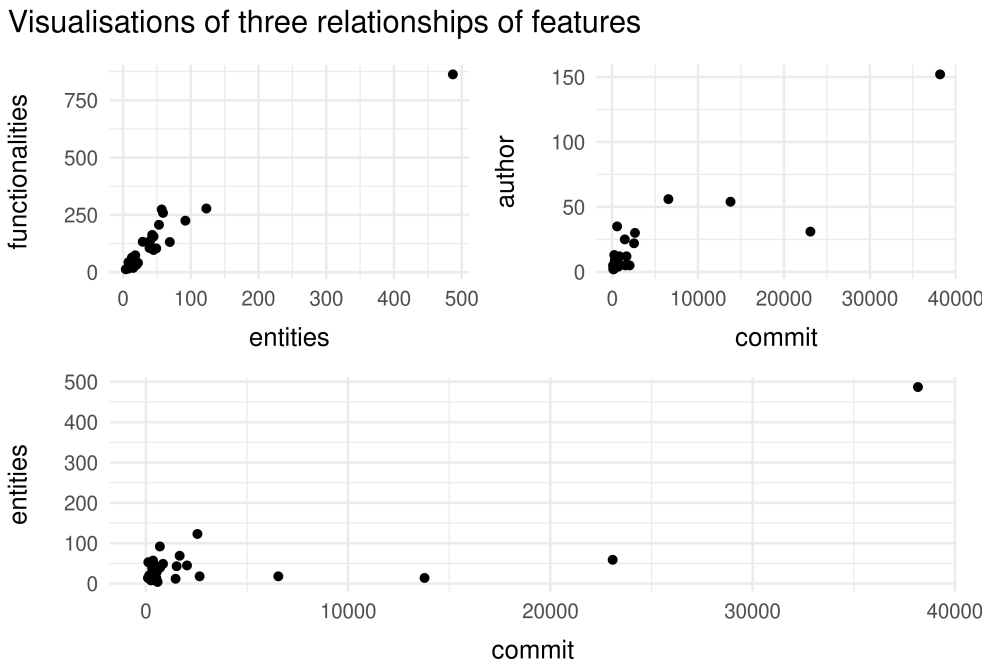

Figure 1: Plots showcasing some relationships of codebases' features: commit, authors, entities, and functionalities.

Overall, the average number of commits is ~ 3578, with a standard deviation of ~ 8373. Considering this, we can say that we have a somewhat diverse distribution of number of commits, which helps our conclusions be more generally applicable. For future reproducibility of our work, table 1 contains the name of the codebases used, their repository url, and the hash of the latest commit we are considering.

Table 1: Information about the considered codebases.

Codebase name	URL (github.com)	HEAD commit hash
Acme-Champions	Agusnez/D05-Acme-Champions	1f903258
APMHome	devhotmail/APMHome	fe087f78
Axon-trader	AxonFramework/Axon-trader	1e987bb1
blended-workflow	socialsoftware/blended-workflow	53082487
cloudstreetmarket.com	alex-bretet/cloudstreetmarket.com	76de2e0d
cloudunit	Treeptik/cloudunit	5079ab85
echo	cardinal76/echo	ac40d97e
edition	socialsoftware/edition	1b2a2b0c
ExtremeWorld	pengchao1989/ExtremeWorld	34d34542
FengHuang	TimYi/FengHuang	17acbe8a
fenixedu-academic	FenixEdu/fenixedu-academic	47eb18f8
hexie	linknabor/hexie	c1d9a85a
irida	phac-nml/irida	b1ea3274
jewelry	masanchezr/jewelry	9dbe6cb8
keta-custom	ketayao/keta-custom	ef8bc0b9
luna	huacha/luna	ab42988e
market-manage	JoleneOL/market-manage	0f3baa4f
quizzes-tutor	socialsoftware/quizzes-tutor	6dcf6684
reddit-app	Baeldung/reddit-app	c7af951a
ruanfan	phyche/ruanfan	ba6e9f45
Skoolie	Kaydub00/Skoolie	8336599f
SoloMusic	jualopmun/SoloMusic	2cf3b54d
splunkwithaws	vinothkrishna15/splunkwithaws	52759335
spring-cloud-gray	SpringCloud/spring-cloud-gray	17cdbd1f
spring-framework-petclinic	spring-petclinic/spring-framework-petclinic	50a219c7
TwitterAutomationWebApp	brianmarey/TwitterAutomationWebApp	dd739539
webofneeds	researchstudio-sat/webofneeds	8fa92d0c
xs2a	adorsys/xs2a	d0642091

2 Results

For each codebase, we create decompositions with 3 to 10 clusters, according to the number of entities it contains: $3 \leq n\_entities < 10 = 3$ clusters; $10 < n\_entities < 20 = 3$, 4, and 5 clusters; $n\_entities \geq 20 =$ 3 to 10 clusters. For each number of clusters, we generate decompositions with varying weights on the six measures: from 0 to 100, with increments of 10. The distribution of the number of generated decompositions can be found in Table 2.

Table 2: The number of generated decompositions across all codebases.

#Entities	#Clusters	#Codebases	#Decompositions
3 to 9	3	4	11982
10 to 19	3 to 5	8	72072
20+	3 to 10	16	384384
Total		**28**	**468438**

2.1 Uniform complexity

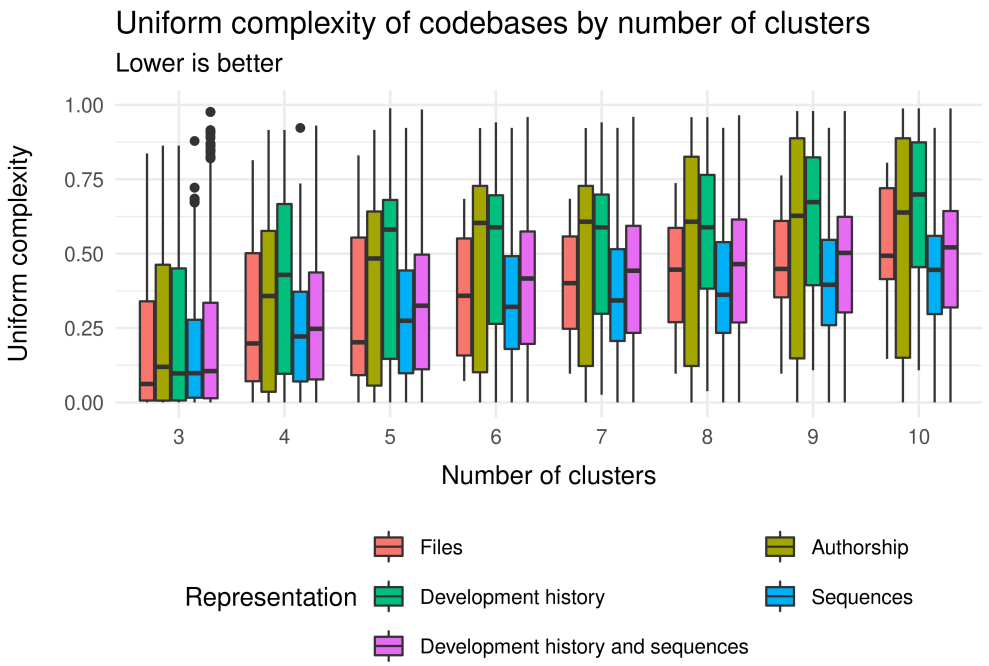

Figure 2: Median uniform complexity of the codebases, per number of clusters and per representation.

We start the analysis of the uniform complexity by comparing the values of all codebases per cluster.

From a visual inspection of Figure 2, it looks like the changes authorship and the file changes representations generate decompositions with higher complexity than the sequence of accesses representation, in all clusters. To confirm this, we apply a Welch T-Test with the following hypotheses:

- H_0 - There are no significant differences between the median uniform complexity of file changes decompositions and sequences of accesses.
- H_1 - The median uniform complexity of file changes decompositions is greater than sequences of accesses.

We only obtain a p-value < 0.05 in the case of 10 clusters, so we can only state with confidence that the median uniform complexity of the file changes decompositions is greater in the case of 10 clusters. Performing a similar test with the authorship representation instead of the file changes representation leads us to a similar conclusion for the case of 4 and 5 clusters (p-values of 0.039 and 0.023, respectively).

The development history representation presents a statistically significant higher median complexity than the sequences of accesses and the file changes, in all clusters. The combined development history and sequences is also statistically significantly higher than the sequences of accesses. All of this indicates that the usage of any development history based representation, generally, generates decompositions with worse complexity than those using the sequences of accesses.

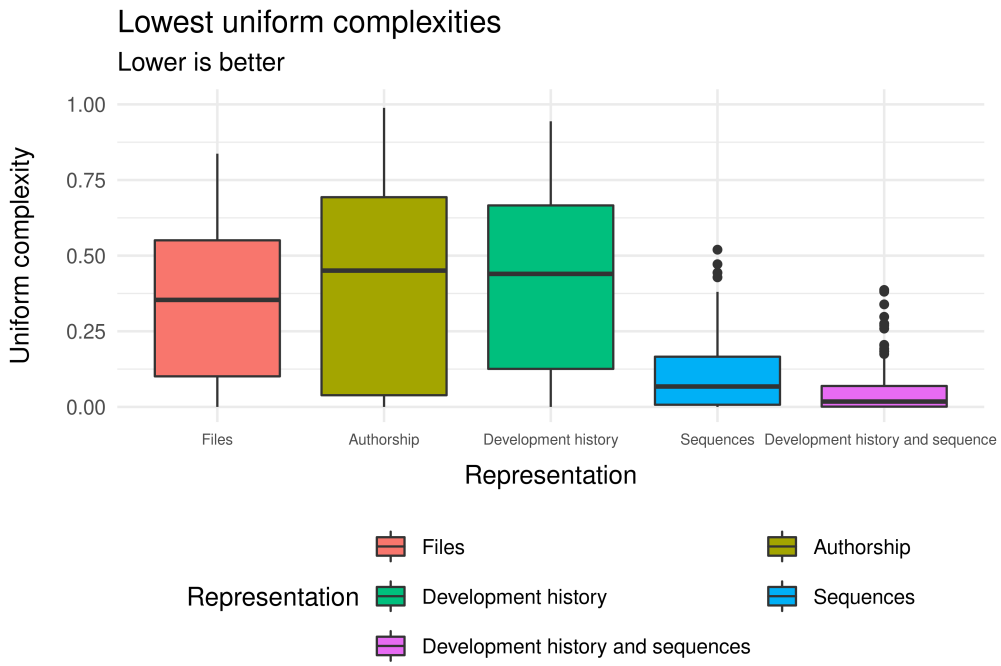

Figure 3: Median uniform complexity of each representation's best uniform complexity decompositions.

The previous analysis showed that, on average, the sequences of accesses representation is better in terms of complexity. However, when selecting the best decomposition of each codebase for each number of clusters, we find that 92.95% of decompositions contain data from a development history representation. This indicates that a detailed analysis on the best decompositions should be made to verify if the same results are obtained.

Figure 3 displays the median uniform complexity of the best decompositions (in terms of complexity) of each representation. What we find is that in this case, the development history and sequences representation is the best, as all other representations have statistically significant greater median. In the case of the development history and sequences representation, 85.9% of the decompositions have a weight in the author measure smaller than 25%. This means that data from the changes authorship representation is generally not given much importance in the best decompositions of the development history and sequences representation. Considering that we cannot state that the development history representation by itself displays higher median than the changes authorship representation, but it does display higher median than the file changes representation, the presence of authorship data seems to often generate decompositions with worse uniform complexity, even in the case of the best decompositions.

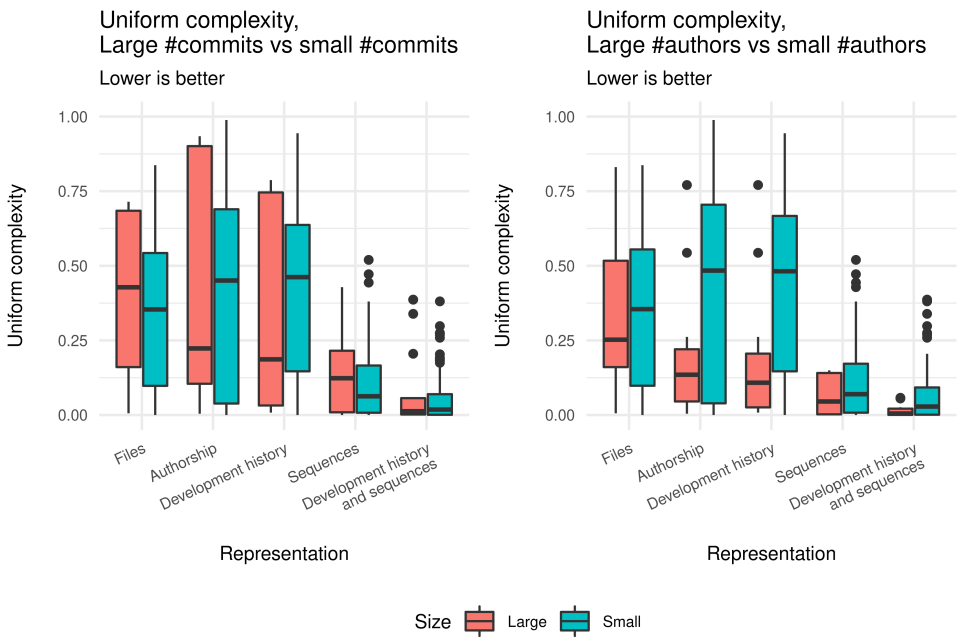

Figure 4: Comparison between the uniform complexity of codebases with a large number of commits with codebases with a small number of commits, and codebases with a large number of authors with a small number of authors. Only the best decompositions were considered.

From the analysis of the best decompositions, we found that combining the sequence of accesses representation with the development history based representation yielded the best results. We also found that the presence of authorship data tended to be detrimental to the complexity value. We considered all codebases, but some codebases have a much larger number of commits and/or authors than the mean. Do the same conclusions as before follow when comparing these large codebases with codebases with less commits and/or authors than the mean?

Figure 4 displays the median uniform complexity of these two groups of codebases. We can say with statistical significance that the median uniform complexity of small codebases in the number of authors is greater than that of large codebases in the number of authors, when comparing the authorship representation for both cases (p-value: 0). We can also state that there are no significant differences between the median uniform complexity of the authorship representation in large codebases to the median uniform complexity of the sequences representation in small codebases, whereas there is a difference if we compare these two representations in small codebases. This is a surprising result, as it suggests the effectiveness of the authorship representation, when compared with sequences, if more authors are present.

In the case of large and small codebases in the number of commits, we cannot state with confidence that there are significant differences between both types of codebases. We find very large quartiles in the boxplots of the development history based decompositions, which means that the uniform complexity obtained is very dependent on the codebase itself.

To summarize: on average, the sequences of accesses representation is better in terms of complexity than any development history based representation; when considering the best decompositions, combining the sequence of accesses representation with the development history representation yields better results, and authorship data worsens the results; codebases with more authors than the mean display better complexity than codebases with less authors than the mean, in the authorship representation, and this representation in large codebases is comparable to the sequences of accesses in smaller codebases.

2.2 Cohesion

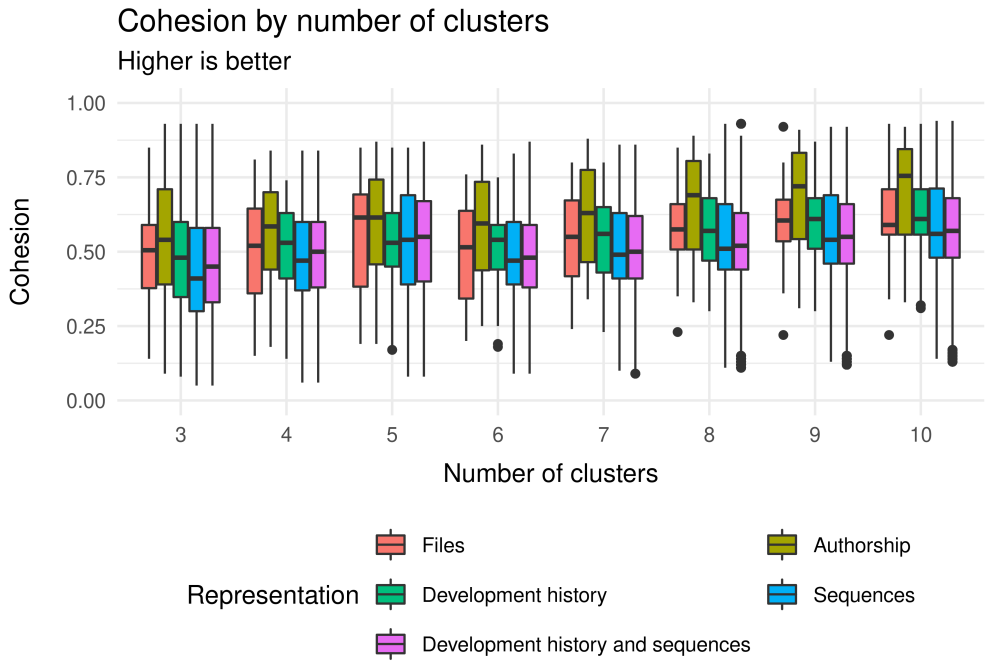

Figure 5: Median cohesion of the codebases, per number of clusters and per representation.

In terms of cohesion, we can visually see in Figure 5 that the sequences representation is very similar to the combined development history and sequences. We can confirm this via a Welch T-test with the following hypotheses:

- H_0 - There are no significant differences between the median cohesion of sequences of accesses decompositions and the development history and sequences.
- H_1 - The median complexity of the development history and sequences decompositions is greater than the sequences of accesses.

We obtain a p-value < 0.05 in the case of 3 and 4 clusters only, so we can only reject H_0 in these two instances. This means that in the majority of clusters, we cannot say that the development history and sequences has better (higher) cohesion than the sequences. We do find, however, that the changes authorship representation has statistically significant higher cohesion than the sequences and the development history and sequences representations, for all clusters except 5. Finally, we cannot confidently state that the file changes has greater median than any other representation, suggesting that it is comparable to them in this metric.

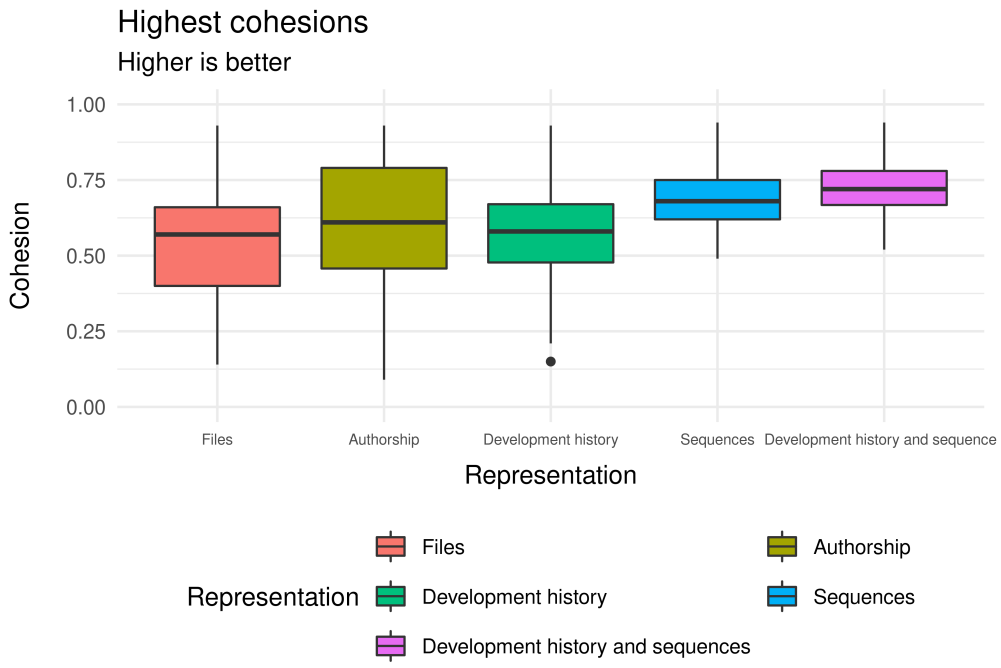

Figure 6: Median cohesion of each representation's best cohesion decompositions.

On average, the previous analysis showed that all representations (other than the changes authorship) are more or less comparable. Following the reasoning from the uniform complexity analysis, we looked at the best decompositions of each codebase and number of clusters, and found that 80.77% have some data from development history based representations. Therefore, it also makes sense to investigate these decompositions in terms of cohesion. The median values for the various representations are displayed in Figure 6.

We find with statistical significance that the best decompositions of the development history and sequences representation have the best values in this metric. We can confidently say that the sequences of accesses is also higher than the three development history based representations. Therefore, combining both data from the development history and data from the sequences of accesses yields best results than just using one of the representations, although the median differences between representations are not as extreme as in the case of uniform complexity.

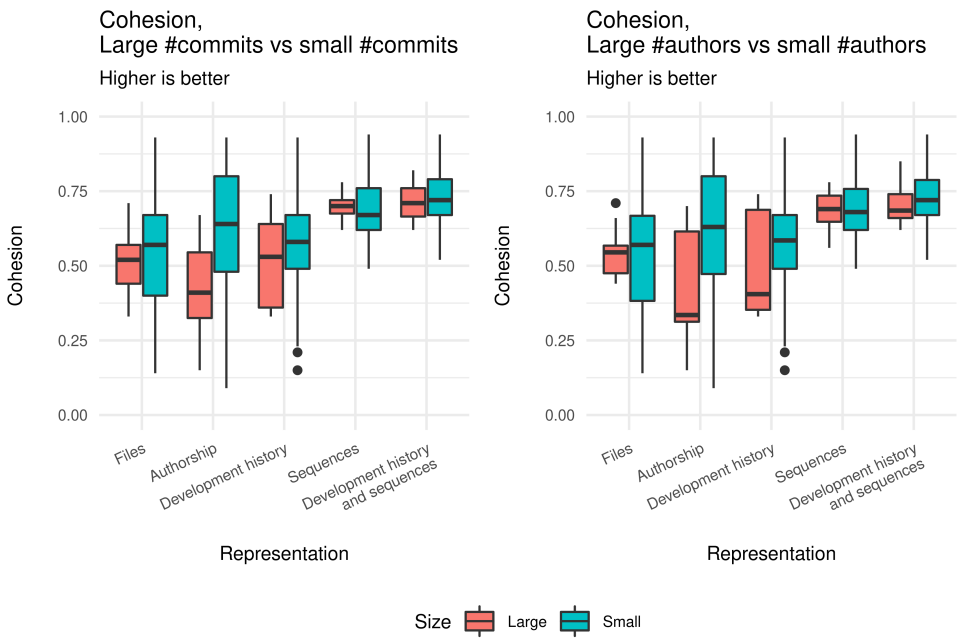

Figure 7: Comparison between the cohesion of codebases with a large number of commits with codebases with a small number of commits, and codebases with a large number of authors with a small number of authors. Only the best decompositions were considered.

Given the poorer performance of the file changes and the changes authorship representations in the best decompositions for all codebases, we now evaluate if there are significant differences between codebases with more commits and/or authors than the mean and codebases with less commits and/or authors than the mean. Figure 7 visualizes the median values of the cohesion, for these two groups, and for each representation.

We can confidently state that the median cohesion of small codebases in the number of commits, in the case of the authorship representation, is greater than the large codebases in the number of commits for the same representation. For the file changes, development history, and development history and sequences, we cannot state that smaller codebases in the number of commits have greater cohesion than the large, and for the sequences of accesses representation, we cannot state that the large codebases display higher median than the smaller.

In the case of codebases with many and few authors, we reach the exact same conclusions. Considering this, the presence of more authors or more commits does not significantly improve cohesion, but less authors seems to generate better decompositions when using the authorship representation. We also find that, both in the case of commits and authors comparisons:

- Comparing the representations of larger codebases, the development history and sequences representation has a statistically significant greater median cohesion than the development history based representations;
- The development history and sequences representation of larger codebases has statistically significant greater median than all other representations of smaller codebases, except the development history and sequences.

This leads us to conclude that combining development history data of large codebases with the sequences of accesses leads to better or comparable results to combining development history data of smaller codebases with sequences of accesses. Just like in the case of uniform complexity, we also see this representation as the best overall.

To summarize: on average, all representations (other than the changes authorship) are more or less comparable; the combined sequence of accesses and development history representation is better than all other representations in the case of the best decompositions; this representation is better in large codebases than in smaller codebases; large codebases, both in number of commits and number of authors, are not better than smaller codebases: smaller codebases in the number of authors display better cohesion in the authorship representation.

2.3 Coupling

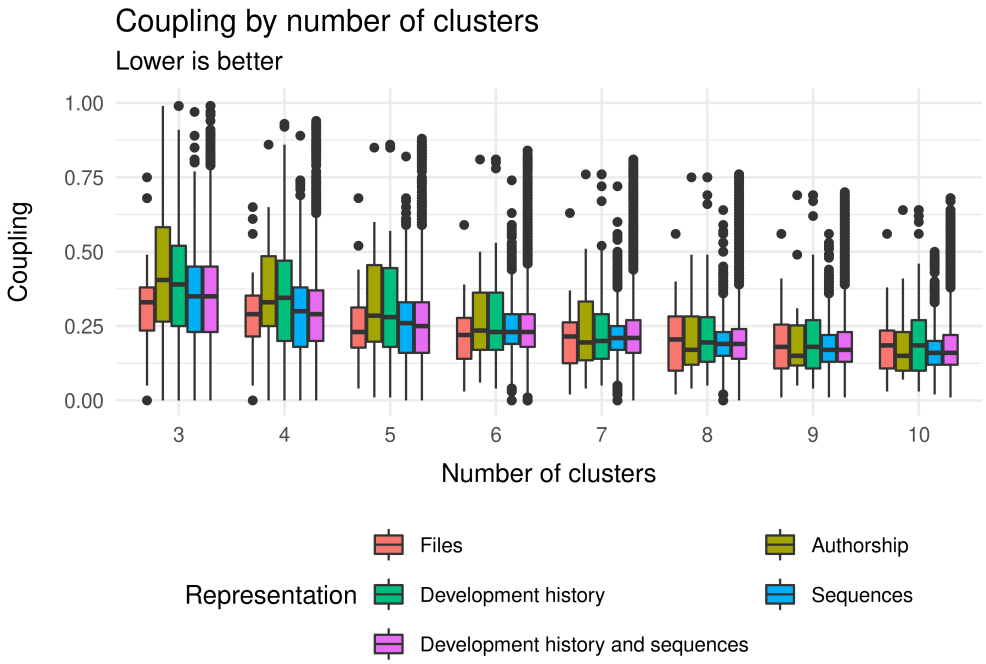

Figure 8: Median coupling of the codebases, per number of clusters and per representation.

Although the coupling looks similar across all representations (see Figure 8), with some variations for some clusters, an analysis with statistical tests reveals that:

- The development history and sequences representation median is higher than the sequences, for all clusters;
- The development history is also higher than the sequences, for all clusters, and is higher than the development history and sequences.

For the remaining combinations of representations, we cannot confidently state than any of them is higher than other consistently. As such, the sequences representation is better than any development history based decomposition for coupling.

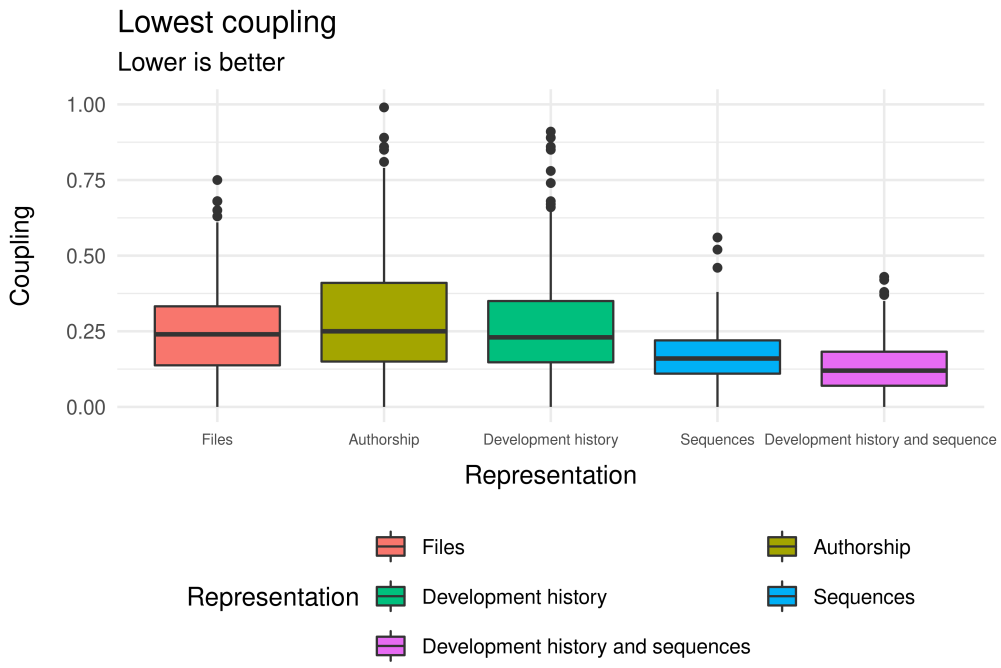

Figure 9: Median coupling of each representation's best coupling decompositions.

Similarly to the previous metrics, 95.51% of the best decompositions in terms of coupling are obtained using the development history and sequences of accesses representation. This warrants investigating the best decompositions of each representation, to evaluate how they compare. A visual comparison is present in Figure 9.

We find that just like the previous metrics, the combination of development history and the sequences of accesses generates the best values, as we always obtain a p-value of 0 when testing if any other representation has higher median. Although we can state, through statistical tests, that the sequences of accesses representation has a higher median value than the development history and sequences representation, we cannot state that it has a higher value than the remaining representations, which does mean that the inclusion of development history data makes the already good sequences of accesses representation even better.

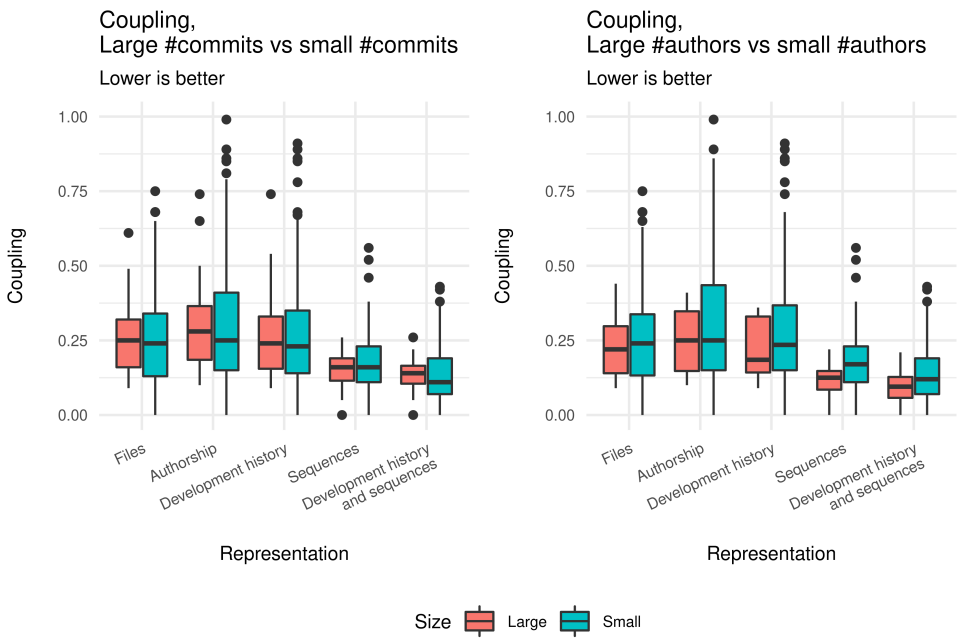

Figure 10: Comparison between the coupling of codebases with a large number of commits with codebases with a small number of commits, and codebases with a large number of authors with a small number of authors. Only the best decompositions were considered.

Since we were not able to state the existence of significant differences between the file changes, changes authorship, and development history representations, it is worth exploring if the presence of more or less commits and/or authors yields different conclusions.

In the case of large and small codebases in the the number of authors, we find that the obtained values in the files and authorship representations are very similar for both categories of codebases, visually (Figure 10) and with statistical significance, but the smaller codebases display worse values in the development history, sequences, and the development history and sequences representations. For large and small codebases in the number of commits, we cannot confidently state that the larger codebases have greater median than the smaller.

To summarize: on average, the sequences of accesses representation performs better than the remaining representations; the best decompositions made with the development history and sequences of accesses representation are better than any other representation; smaller codebases in the number of authors are worse for some representations, but no differences were found between larger and smaller codebases in the number of commits.

2.4 Team size reduction ratio

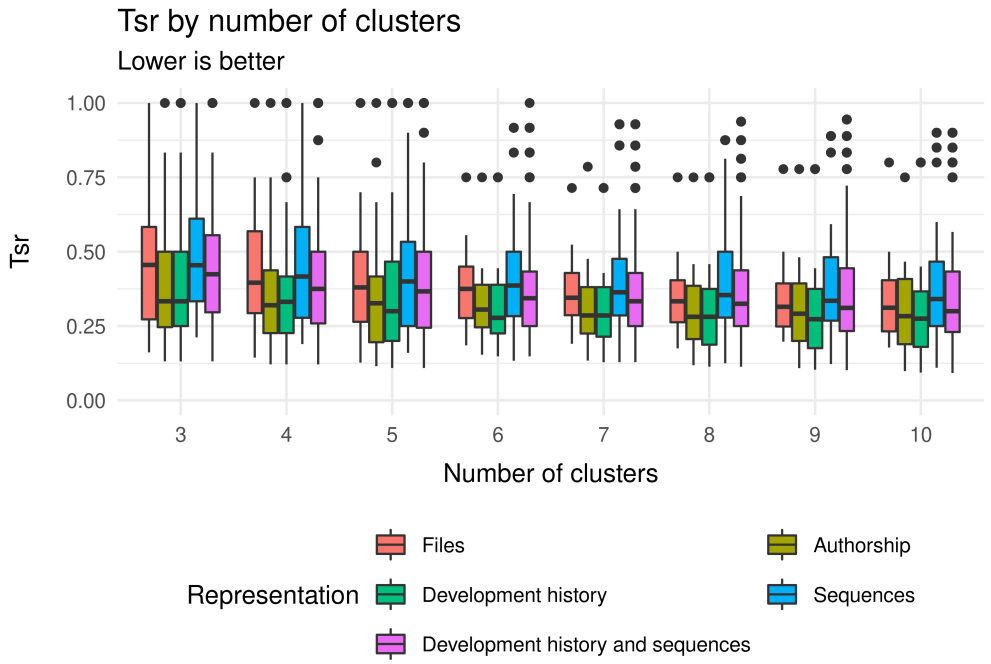

Figure 11: Median tsr of the codebases, per number of clusters and per representation.

In this metric, we find more consistent results across all clusters, which can be seen in Figure 11. Similarly to the cohesion, including data from the changes authorship representation produces better results than not including it. The changes authorship representation is significantly similar to the development history based representation, indicating that the usage of authorship data has a great influence when combined with the file changes data. As this metric evaluates the reduction in team size per microservice, when compared with the monolith's original team, it makes sense that decompositions made with author data perform better. However, it's interesting to see that the decompositions made with the sequences of accesses representation, despite having no data regarding authors, display an acceptable median value under 0.5, and the quartiles in decompositions with 6 or more clusters are all under 0.5 as well.

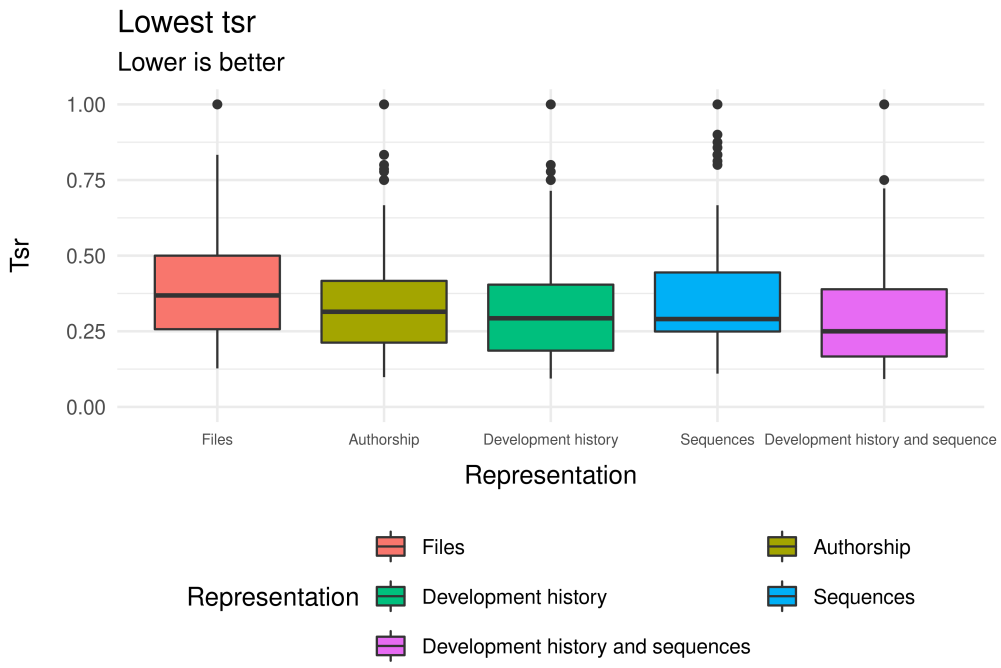

Figure 12: Median tsr of each representation's best tsr decompositions.

In total, 95.51% of the best decompositions in terms of tsr were achieved with the development history and sequences representation, which is an indication that, once again, we should compare the remaining representations in this context.

We cannot state that the development history representation has a higher median than the development history and sequence of accesses representation, highlighting that there is little benefit to include the sequences of accesses. Indeed, around 74% of the best development history and sequences of accesses decompositions have a sum of weights for the author and commit similarity measures greater than 50.

We can state that the authorship representation has higher median than the development history and sequences representation, but cannot state that it is higher for any other representation. We also cannot state that the file changes representation has higher median than the sequences of accesses, but it does have higher median than the remaining representations.

All in all, the development history and sequences representation is still the best, as all other representations are significantly higher. The sequences of accesses representation, despite having no author data, still displays comparable results to the authorship representation, which has only author data.

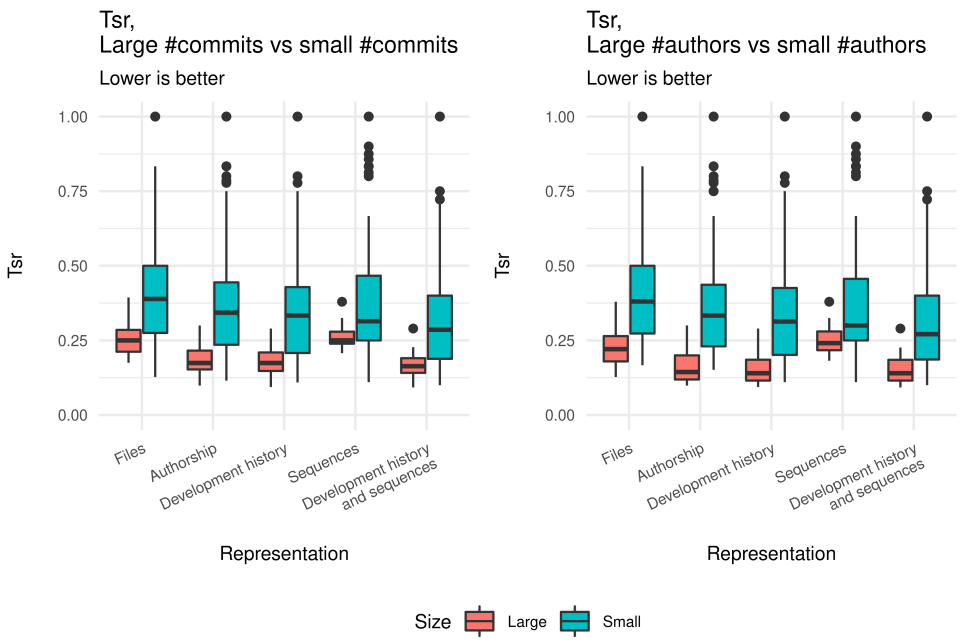

Figure 13: Comparison between the team size reduction of codebases with a large number of commits with codebases with a small number of commits, and codebases with a large number of authors with a small number of authors. Only the best decompositions were considered.

Considering that we cannot state the existence of differences between the changes authorship and the sequences of accesses representations, it is worth checking if the presence of more authors changes this.

Comparing large and small codebases in the number of authors, we can say that the tsr of the smaller codebases in the number of authors is higher, and therefore worse, than the large ones in all representations. Statistical tests reveal that there are no significant differences between the authorship, development history, and the combined development history and sequences representations for the small codebases, highlighting the effect that the changes authorship data has in these representations. Similarly to the uniform complexity, the development history based representations of large codebases perform better than the sequence of accesses of small codebases, whereas they perform worse in the case of smaller codebases, so the presence of more authors does change the conclusions of the previous analysis of the best decompositions.

Large codebases in the number of commits also display improved tsr median values across all representations, when compared with small codebases in the number of commits.

To summarize: on average, decompositions which include changes authorship data perform better than those that don't; in the best decompositions, the development history and sequences of accesses representation does not present siginificant differences to the development history representation, and the sequences of accesses representation is not significantly different from the authorship representation, despite having no author data; the presence of more commits and/or more authors significantly improves this metric in all representations.

2.5 Combined

Figure 14: Median combined of the codebases, per number of clusters and per representation.

This metric offers an overview of all metrics and the trade-offs made in the decomposition, by combining the previous metrics into a single value. There is not a single type of representation that can be said to be the best for all clusters and metrics - from what we've covered so far, some representations excel at some metrics more than others. Therefore, it is natural that most decompositions display median values in each type of representation of around 0.4, with quartiles going up to 0.5. This can be seen in Figure 14.

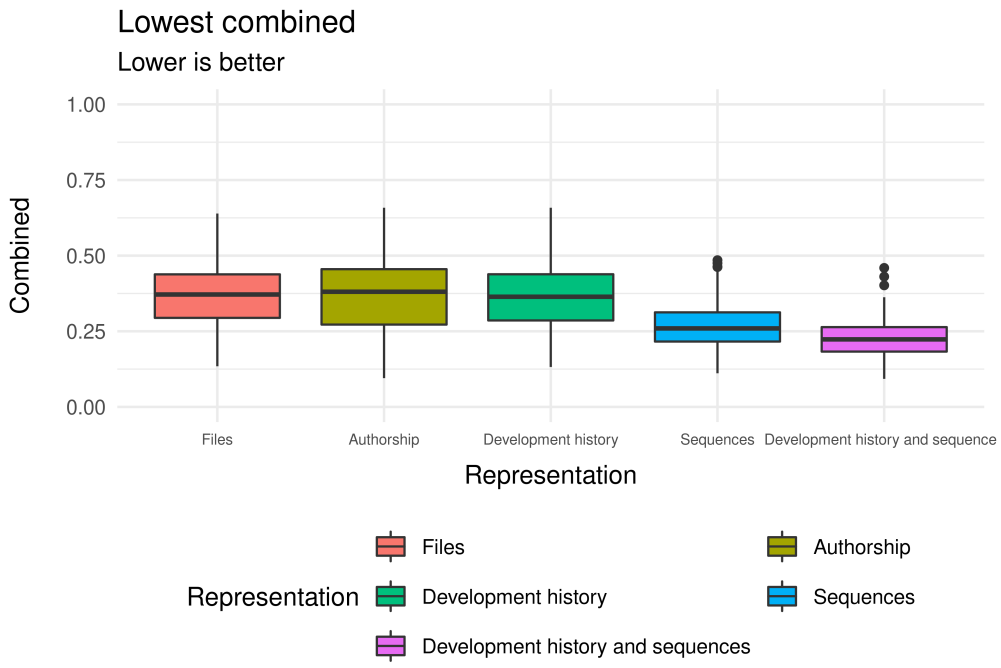

Figure 15: Median combined of each representation's best combined decompositions.

Since we cannot state that, on average, a single representation is better than the others, we can look at the best decompositions (in Figure 15) to get another perspective.

The analysis of this plot confirms the analysis of the previous metrics: the best decompositions of the development history and sequences representation are the best across all metrics, as all others representations display statistically significant higher values. The sequences representation, due to the better values in uniform complexity, cohesion, and coupling, is also good, as it has a higher median than the development history and sequences representation, but the other representations have a higher median the the sequences of accesses. Between the file changes, authorship, and development history representations, we cannot state that any of them is higher than another, which highlights the better performance of some representations in some metrics, but worse performance in others.

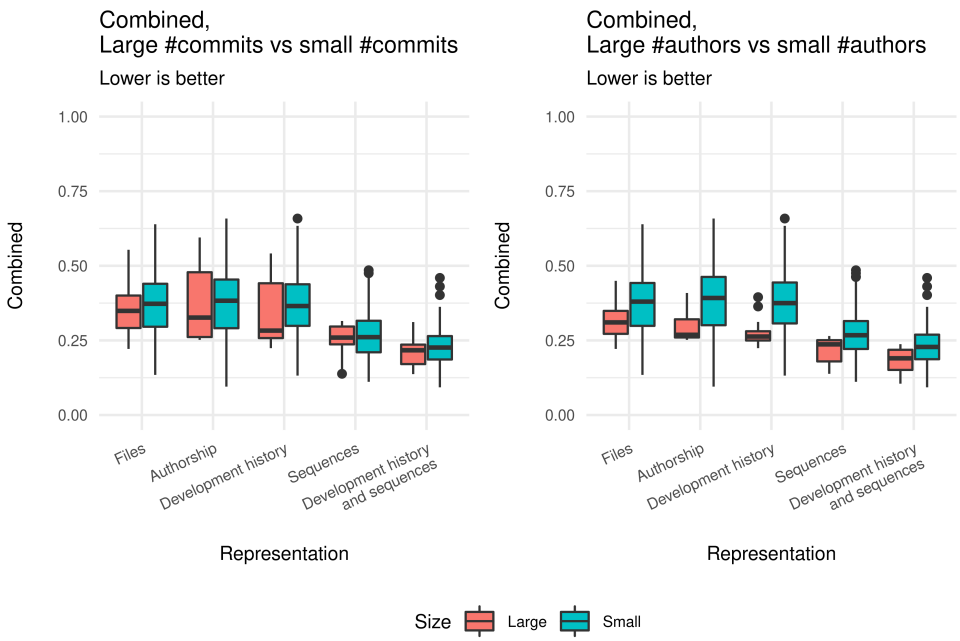

Figure 16: Comparison between the combined of codebases with a large number of commits with codebases with a small number of commits, and codebases with a large number of authors with a small number of authors. Only the best decompositions were considered.

Considering the similarity between the file changes, changes authorship, and development history representations, we can try to find if there are any differences when considering large and small codebases in the number of commits and/or authors. Figure 16 contains a visualization of this comparison.

The good performance of the larger codebases in the number of authors for previous metrics explains the improvement of the combined metric, as the representations of smaller codebases have statistically significant greater median values than the large codebases. For the case of commits, we cannot state that the smaller codebases display larger values in any representations, which means that overall, the quality of codebases with more commits is comparable to the quality of codebases with less commits.

To summarize: on average, we cannot state that any representation is better than others; the development history and sequences of accesses is the best representation when analyzing the best decompositions, which is expected considering that this representation was consistently the best in the previous metrics; large codebases in the number of authors are better than smaller, whilst no significant differences were found betweeen large codebases in the number of commits and small codebases in the number of commits.

3 Evaluation conclusions

We were able to properly compare the various representations in previous sections, through visual analysis and statistical analysis. The median quality metrics values across all codebases and clusters provided us with an overview of the most common scenario. But this is a limited view, as it does not fully represent what happens if an ideal decomposition is found. To fix that, we also analyzed the best decompositions of each metric and representation, where often different results were obtained. Additionally, we often could not find significant differences between the development history based representations, so we separated the best decompositions according to the number of commits and authors of their codebases, which gave us better insights in some situations.

This whole analysis provided us with several conclusions that allow us to answer the research question of this thesis, focused around how the sequences of accesses representation compares with development history based representations:

- On average, the sequence of accesses representation is better in the case of complexity and coupling. For cohesion, no significant differences were found between representations, with the exception of the changes authorship representation, which performs worse than all others. On the other hand, representations with change authorship data, like the changes authorship, development history, and development history and sequences, perform better than the sequence of accesses for tsr, although we have to highlight the still acceptable performance of the latter. These trade-offs are captured by the combined metric and confirmed through statistical tests, as we cannot state that any representation is better than the others at any number of clusters.
- When looking at the best decompositions, the combined development history and sequence of accesses representation yielded the best values for all metrics. A notable exception is in the tsr metric, where we could not state with statistical significance that this representation is better than the development history representation. Nevertheless, these conclusions give support to the notion that the presence of more data from different sources improves results. However, considering that this does not happen when looking at the vast majority of decompositions, it means that obtaining good results when using data from the development history is very dependent on choosing the ideal weights when creating the decomposition.
- Through the comparison between large and small codebases in the number of commits and/or authors, we cannot state that the complexity of the changes authorship representation of large codebases in the number of authors is higher than the sequences of accesses representation of small codebases in the number of authors. Regarding cohesion and coupling, we found that the presence of more commits or authors does not improve results. On the other hand, more commits or authors does significantly improve the tsr values, as any representation of larger codebases has a better median than the sequences of accesses representation of smaller codebases.

4 Threats to validity

Out of all decompositions, we find that 0.37% were made exclusively with data from the development history, and 9.52% exclusively with data from the sequences of accesses. The remaining 90.11% were made with combined data. This is a consequence of using four measures related to the sequences of accesses, but only two related to the development history. To ensure this does not affect our findings, we opted to use a statistical test that performs well even comparing groups with different sample sizes, and do not rely only on boxplots to draw conclusions. Additionally, note that these differences only applied for the first analysis, where we consider to all the decompositions, which was rather inconclusive. For all the other analyses, the best decompositions were chosen and so, we have only one decomposition per representation and number of clusters.

Not all repositories have a clean and linear history, with some presenting many branches, refactors, and merges. This affects the detection and processing of deletes and renames, which makes development history based decompositions less efficient. Nevertheless, we obtained good results for the development history representations.

We found that sometimes, files presented an `ADD` or `MODIFY` change event after a `DELETE` event. In some situations, this means we could be considering two distinct files as the same one, if they happened to have the same filename and one of them was deleted before the other was added. However, in all cases we found, the files still existed in the latest repository snapshot and did correspond to the same file that was deleted. Considering that this situation is unlikely, and the existence of a `DELETE` event after an `ADD` or `MODIFY` change event usually occurs due to merges, we opted to still consider these files in our analysis, and we don't discard them.

Our data collection approach was to gather data about all `.java` files across all commits, and then discard non domain entities files only in the decomposition phase. An alternative would be to filter all commits and select those where only domain entities were changed. Our approach is richer, as we have more data available and are not deleting potentially useful relationships between files.

We adapted the logical coupling and the contributor coupling measures from [2], by considering a fraction rather than an absolute value. This was made to facilitate the integration of other measures without much experimentation on the ideal weights that would be required if absolute values were considered.

References

[1] S. Santos and A. R. Silva, "Microservices identification in monolith systems: Functionality redesign complexity and evaluation of similarity measures," *Journal of Web Engineering*, Aug. 2022, doi: 10.13052/jwe1540-9589.2158.

[2] G. Mazlami, J. Cito, and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures," in *2017 IEEE International Conference on Web Services (ICWS)*, Jun. 2017, pp. 524–531. doi: 10.1109/ICWS.2017.61.