

Experiences in Replicating an Experiment on Comparing Static and Dynamic Coupling Metrics

Richard Müller¹, Dirk Mahler² and Christopher Klinkmüller³

¹*ipoque GmbH, Leipzig, Germany*

²*BUSCHMAIS GbR, Dresden, Germany*

³*CSIRO Data61, Sydney, Australia*

Abstract

In software engineering, coupling metrics are used to assess the quality of a software system's architecture, especially its maintainability and understandability. On an abstract level, two types of coupling metrics can be distinguished: static metrics are solely based on source and/or byte code, and dynamic metrics also take observed run-time behavior into account. While both metric types complement each other, a recent study by Schnoor and Hasselbring suggests that these two types are correlated. This observation indicates that to a certain degree both metric types encode redundant information. In this paper, we replicate the original experiment using the same data but a different tool set. Our successful replication hence substantiates the original findings. Moreover, the summary of our experience provides valuable insights to researchers who want to ensure reproducibility and replicability of their experiments. Following open science principles, we publish all data and scripts online.

Keywords

Software Metrics, Monitoring, Dynamic Analysis, Static Analysis, Open Science, Replication

1. Introduction

In software engineering, a common design principle for improving the quality of a system architecture, especially its *maintainability* and *understandability*, is to ensure a high cohesion within modules and a low coupling between modules [1, 2]. *Coupling metrics* serve as indicators for the degree to which this design principle is adhered to and thus for the quality of the architecture. A fundamental metric is the *coupling degree* of a module which measures the number of connections between a module and other system modules [3], for example, between classes or packages. It can be restricted to certain kinds of connections, such as method calls, types of member variables, or types of thrown exceptions. Depending on whether the connections were derived via static analysis of source and/or byte code, or via dynamic analysis of monitoring logs, coupling degrees (or metrics) are referred to as *static* or *dynamic*, respectively.

Schnoor and Hasselbring [3] recently investigated the extent to which the information provided by dynamic coupling metrics complements the information captured by static metrics, and vice versa. To this end, they empirically analyzed coupling orders of modules for a specific

SSP'21: Symposium on Software Performance, November 09–10, 2021, Leipzig, Germany


✉ richard.mueller@rohde-schwarz.com (R. Müller); dirk.mahler@buschmais.com (D. Mahler);

christopher.klinkmueller@data61.csiro.au (C. Klinkmüller)

ORCID 0000-0001-6730-4082 (R. Müller); 0000-0002-5926-2238 (C. Klinkmüller)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

software system. That is, they compared rankings of system’s modules obtained by ordering the modules with respect to a variety of static and dynamic coupling degrees. The observation that rankings based on static coupling degrees are not statistically independent from rankings based on dynamic coupling degrees led Schnoor and Hasselbring to the conclusion that the information provided by both types of coupling degrees is related.

In this paper, we *replicate* the experiment conducted by Schnoor and Hasselbring [3] to verify the previously observed findings. While we rely on the input data provided by the authors and follow their procedure for computing and comparing the coupling degrees, we apply a different tool set to process and analyze the data. This change implies that we go beyond a reproduction of the original experiment¹. Additionally, we report our experiences from the replication study, explicating challenges that we faced and that are of interest to researchers wanting to ensure reproducibility and replicability of their experiments.

In the remainder of the paper, we summarize the original experiment in Section 2. We then outline our replication setup and results in Section 3 and discuss our experiences in Section 4. Finally, Section 5 concludes the paper. Following open science principles [4], all scripts used in this replication study are publicly available and can be executed online².

2. Original Experiment

Schnoor and Hasselbring [3] empirically examined the extent to which different coupling degrees provide similar information. For this purpose, they investigated Atlassian Jira³, a commercial issue tracking system, in a series of four experiments. Each experiment spanned a period of four weeks in which the system was actively used by students participating in a mandatory programming course and in which data reflecting its runtime behavior was collected. Note that the experiments relied on different versions of Jira, as Schnoor and Hasselbring updated their Jira installation over time. However, while they used different versions across the series of experiments, within each experiment the same version was applied. As a consequence, the system’s architecture and runtime behavior varied, albeit slightly, across all experiments. Table 1 briefly summarizes the main characteristics of the experiments.

For each experiment, the same analytical procedure was followed. First, the static and dynamic dependency graphs based on method calls were created for each experiment. The static dependency graph was derived from a byte code scan of the corresponding Jira version using BCEL⁴. By contrast, the dynamic dependency graph was derived from monitoring data obtained at runtime during the experiment run with the monitoring tool Kieker [5]. For each experiment the monitoring data is provided as a publicly available dataset^{5,6,7,8}. Each dataset

¹We follow the terminology defined by the Association for Computing Machinery in <https://www.acm.org/publications/policies/artifact-review-and-badging-current>.

²<https://github.com/softvis-research/coupling-metrics-replication>

³<https://www.atlassian.com/de/software/jira>

⁴<https://commons.apache.org/proper/commons-bcel/>

⁵<https://doi.org/10.5281/zenodo.3648094> (experiment 1)

⁶<https://doi.org/10.5281/zenodo.3648228> (experiment 2)

⁷<https://doi.org/10.5281/zenodo.3648240> (experiment 3)

⁸<https://doi.org/10.5281/zenodo.3648269> (experiment 4)

Table 1

Numbers of users, monitored calls, and used Jira versions of the four experiments [3].

#	Date	Users	Method Calls	Jira version
1	February 2017	19	196,442,044	7.3.0
2	September 2017	48	854,657,027	7.4.3
3	February 2018	16	475,357,185	7.7.1
4	September 2018	58	2,409,688,701	7.7.1

yields compressed binary files (`bin.xz`) and a kieker file (`.map`) that contains the encrypted fully qualified class names.

Next, the coupling degrees were derived from the dependency graphs. The nodes of this graph are program modules that are either *classes* or *packages*. The edges are aggregated *call* relationships between the modules which can have weights representing the number of calls. Depending on whether the coupling metrics take the weights into account, they are referred to as *weighted* or *unweighted*. Please note that weighted metrics are omitted in the static dependency graph as byte code is used and compiler optimizations may produce different results. Furthermore, the direction of the calls is distinguished into outgoing and incoming calls referred to as *import* and *export* coupling degree of a program module. The sum of import and export coupling is referred to as *combined* coupling.

Finally, the differences of the coupling metrics were studied by comparing the ranking obtained by ordering the program modules by their coupling degree using the Kendall-Tau distance [6]. Values smaller than 0.5 indicate that the orders are closer together than expected from two random orders. Distance values larger than 0.5 indicate the opposite. Considering the above-mentioned measurements this results in 18 comparisons. The results were presented using the following triple $\alpha : \beta_1 \leftrightarrow \beta_2$, where

- α is *c* or *p* expressing **class** or **package** coupling,
- β_1 is *s* or *u* expressing whether the left-hand side analysis is **static** or (dynamic) **unweighted**,
- β_2 is *u* or *w* expressing whether the right-hand side analysis is (dynamic) **unweighted** or (dynamic) **weighted**.

Schnoor and Hasselbring observed that the distance values for all pairs of compared rankings were smaller than 0.5. Table 2 shows the obtained distance values for all four experiments. These observations led the authors to the conclusion that coupling metrics obtained from static and dynamic analysis encode similar information.

3. Replication

The major steps of the replication process with corresponding inputs and outputs are summarized in Figure 1. First, we used jQAssistant⁹ and its scanner plugins for Kieker [7, 8] and for Java byte code [9] to process the Kieker traces provided by Schnoor and Hasselbring as well as

⁹<https://jqassistant.org/>

Table 2

Coupling analysis results of all four original experiments [3] including the differences (+/-) of the replication study.

(a) Experiment 1.						
	$c : s \leftrightarrow u$	$c : s \leftrightarrow w$	$c : u \leftrightarrow w$	$p : s \leftrightarrow u$	$p : s \leftrightarrow w$	$p : u \leftrightarrow w$
import	0.31+0.01	0.36+0.01	0.13-0.01	0.33	0.36-0.01	0.08
export	0.41	0.41-0.01	0.24	0.30	0.32-0.01	0.21-0.01
combined	0.35	0.41-0.01	0.29-0.01	0.29	0.33-0.01	0.23-0.01
average	0.35	0.39	0.22-0.01	0.31	0.33	0.17
(b) Experiment 2.						
	$c : s \leftrightarrow u$	$c : s \leftrightarrow w$	$c : u \leftrightarrow w$	$p : s \leftrightarrow u$	$p : s \leftrightarrow w$	$p : u \leftrightarrow w$
import	0.30+0.02	0.36+0.01	0.14	0.31	0.35	0.09-0.01
export	0.41	0.43-0.01	0.26-0.01	0.30	0.33	0.22-0.01
combined	0.34+0.01	0.41-0.01	0.31-0.01	0.28	0.33-0.01	0.23
average	0.35+0.01	0.40	0.24-0.01	0.30	0.33	0.18
(c) Experiment 3.						
	$c : s \leftrightarrow u$	$c : s \leftrightarrow w$	$c : u \leftrightarrow w$	$p : s \leftrightarrow u$	$p : s \leftrightarrow w$	$p : u \leftrightarrow w$
import	0.38	0.42+0.01	0.12	0.37	0.39	0.06
export	0.38	0.40	0.22	0.28	0.31	0.20
combined	0.36	0.40+0.01	0.28	0.30	0.33	0.23+0.01
average	0.37	0.41	0.21	0.32	0.35	0.17
(d) Experiment 4.						
	$c : s \leftrightarrow u$	$c : s \leftrightarrow w$	$c : u \leftrightarrow w$	$p : s \leftrightarrow u$	$p : s \leftrightarrow w$	$p : u \leftrightarrow w$
import	0.37	0.42	0.12	0.36	0.39	0.06
export	0.38	0.40	0.23	0.28	0.32	0.20
combined	0.35	0.40+0.01	0.29	0.30	0.33	0.24
average	0.37	0.41	0.21	0.31	0.35	0.17

Jira's Java byte code. This resulted in static and dynamic dependency graphs that contained call relationships at method-level. We stored these graphs in a Neo4j database¹⁰. Second, we aggregated the method calls at class- and package-level using custom Neo4j Cypher queries specified as jQAssistant concepts [9]. Third, we queried the graphs using Cypher to calculate the import, export, and combined coupling degrees for each module. Finally, we compared the coupling orders of the modules using the Kendall-Tau distance.

We implemented steps 1 and 2 as batch scripts and steps 3 and 4 in a Jupyter notebook¹¹. The complete replication package including the batch scripts, the Jupyter notebook, and instructions

¹⁰<https://neo4j.com/>

¹¹<https://jupyter.org/>

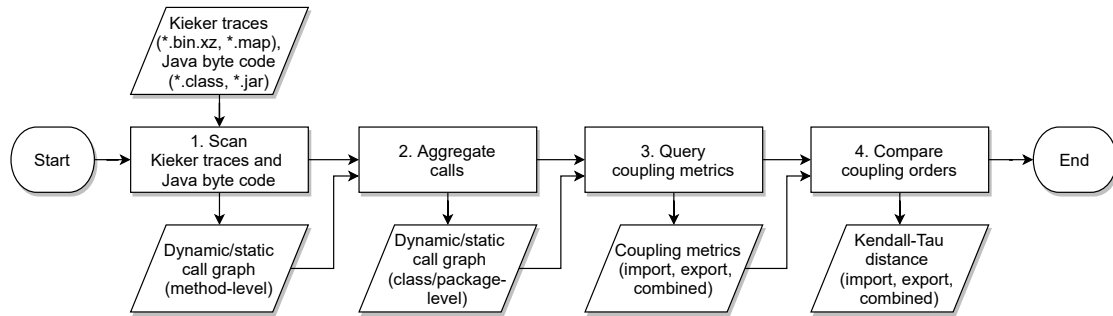


Figure 1: Flowchart of the replication process.

are provided in a GitHub repository and can be executed online, see Footnote 2.

The results of the replication study are also shown in Table 2. The slight differences between the originally reported and the replicated values are probably due to the following two circumstances. First, for unknown reasons a few classes that we discovered in the static dependency analysis were not included in the static dependency graphs in the original experiment. However, in the original experiment they were included in dynamic dependency graphs. Second, the kieker files of experiment 1 and 2 contain an empty key-value pair in the the second line (\$1=). Here, a class might be missing in the published datasets that was contained in the original experiments.

4. Discussion

Using our setup we were able to replicate the Kendall-Tau distances from the original experiment and thus to substantiate the findings by Schnoor and Hasselbring. However, we faced some challenges that exacerbated the replication. As we could eventually resolve all of them, not least due to the help of the authors, we share our experiences in the following, hoping to contribute to a better understanding of the level of detail with which experiments must be reported and documented, in order to warrant their successful reproduction or replication.

We encountered two challenges regarding the data that was available for the replication. First, in addition to different Jira versions, Atlassian also distinguishes between different Jira variants, for example, Jira Core and Jira Software. As information regarding the latter was not specified in [3], it was unclear which variant served as the basis for the static analysis. Second, the names for packages, classes, and methods were pseudonymized in the publicly available monitoring data. In more detail, the fully qualified Java names were replaced with versions in which each of a name’s components was replaced with its hash. For example, a fully qualified class name like *package.subpackage.class\$innerclass* would be represented as *hash(package).hash(subpackage).hash(class\$innerclass)*. On the one hand, we were not able to distinguish between class and inner class calls. On the other hand, the encryption also affected the creation of module rankings, because the clear names were used as a secondary sorting criterion in the original experiment. We thus had to decrypt the pseudonymized names in the monitoring data and replaced them with the clear names from the Java byte code.

Furthermore, a few challenges occurred during the analysis of the dependency graphs or module rankings, respectively. First, we tested different Python implementations of the Kendall-Tau distance. All of them produced slightly different results, none of which were equal to the results from the original experiment. In the end, we implemented our own calculation for the Kendall-Tau distance. Second, not all call relationships between classes and packages were actually considered by Schnoor and Hasselbring. Here, we found that the selection criteria specified in [3] did not yield the desired results. Yet, with the help of the authors, we were able to determine that lambda method calls indeed needed to be excluded, but constructor calls had to be included.

Lastly, we identified a mistake in the documentation of the results of the original experiment. In particular, Table 10 in the original paper shows the average export coupling degrees in the four experiments. In case of the static dependency graphs these values are calculated using the static weighted dependency graph. However, with the static analysis only the unweighted coupling degrees were used in the original experiment. As we took the reported averages in Table 10 to confirm the correctness of our replication setup, we first suspected a problem in our setup, leading to significant verification efforts.

In summary, we encountered a variety of challenges related to the available data, the applied method, and the documented results. We would like to emphasize that none of the identified challenges led to significant differences between the original and our replicated results. However, the investigation and rectification of these issues required time and effort, especially considering that their effects influenced each other and that fixing one issue often revealed a new one.

5. Conclusion

We successfully replicated the experiment from Schnoor and Hasselbring [3] which empirically investigated differences between static and dynamic coupling metrics. Hence, we could substantiate the authors' finding that static and dynamic coupling degrees are not statistically independent. Furthermore, we discussed our experiences and, following open science principles, made all scripts of the replication study available in a reproduction package that can be executed online, see Footnote 2.

In general, researchers who strive to ensure reproducibility or replicability should check whether the empirical data, the applied methods, and the obtained results are sufficiently well described in the paper and/or documented elsewhere. In this regard, we greatly benefited from the help of the authors and hence encourage researchers who plan a replication study to contact the authors of the original study as early as possible. A specific recommendation arising from our replication study is that researchers who pseudonomize or encrypt (parts of the) data should review and explicate how this affects the reproduction or replication of their results.

Acknowledgments

We would like to thank the authors of the original experiment, Henning Schnoor and Wilhelm Hasselbring, for publishing their data and for their great support with solving the challenges of this replication study.

References

- [1] I. Candela, G. Bavota, B. Russo, R. Oliveto, Using Cohesion and Coupling for Software Remodularization: Is It Enough?, *ACM Trans. Softw. Eng. Methodol.* 25 (2016). doi:10.1145/2928268.
- [2] J. Bogner, S. Wagner, A. Zimmermann, Automatically Measuring the Maintainability of Service- and Microservice-Based Systems: A Literature Review, in: *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, IWSM Mensura '17*, Association for Computing Machinery, New York, NY, USA, 2017, pp. 107–115. doi:10.1145/3143434.3143443.
- [3] H. Schnoor, W. Hasselbring, Comparing Static and Dynamic Weighted Software Coupling Metrics, *Computers* 9 (2020) 24. doi:10.3390/computers9020024.
- [4] D. Mendez, D. Graziotin, S. Wagner, H. Seibold, Open Science in Software Engineering, in: M. Felderer, G. H. Travassos (Eds.), *Contemporary Empirical Methods in Software Engineering*, Springer International Publishing, Cham, 2020, pp. 477–501. doi:10.1007/978-3-030-32489-6_17.
- [5] W. Hasselbring, A. van Hoorn, Kieker: A monitoring framework for software engineering research, *Software Impacts* 5 (2020) 1–5. doi:10.1016/j.simpa.2020.100019.
- [6] L. Briand, K. E. Emam, S. Morasca, On the application of measurement theory in software engineering, *Empirical Software Engineering* 1 (1996) 61–88. doi:10.1007/BF00125812.
- [7] R. Müller, M. Fischer, Graph-Based Analysis and Visualization of Software Traces, in: *10th Symposium on Software Performance*, Würzburg, Germany, 2019.
- [8] R. Müller, T. Stempel, Graph-Based Performance Analysis at System- and Application-Level, in: *11th Symposium on Software Performance*, Leipzig, Germany, 2020.
- [9] R. Müller, D. Mahler, M. Hunger, J. Nerche, M. Harrer, Towards an Open Source Stack to Create a Unified Data Source for Software Analysis and Visualization, in: *6th IEEE Working Conference on Software Visualization*, IEEE, Madrid, Spain, 2018. doi:10.1109/VISSOFT.2018.00019.