





DemeterWatch: A Java tool to detect Law of Demeter violations in Java collections

Juan Pablo P. de Aquino ^{1*}, José Fernando de M. Firmino ^{2*}, Diogo D. Moreira^{1*}, and Ricardo de S. Job^{1*}

1 Instituto Federal de Educação Ciência e Tecnologia da Paraíba - IFPB, Brazil 2 Universidade Federal da Paraíba - UFPB, Brazil  Corresponding author * These authors contributed equally.

DOI: [10.21105/joss.07308](https://doi.org/10.21105/joss.07308)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Vissarion Fisikopoulos](#)  

Reviewers:

- [@saaikrishnan](#)
- [@louiseadennis](#)

Submitted: 10 September 2024

Published: 15 March 2025

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

Object-oriented languages brought important concepts and concerns to the software industry, such as inheritance, polymorphism, and coupling. Since the 1970s, these concepts have been studied, and in particular, coupling has emerged as an important mechanism that allows components to keep the details of their construction restricted ([Parnas, 1972](#)). In this sense, when an object's encapsulation is compromised, its implementation becomes exposed, and other parts of the code can directly alter its behavior, reducing the cohesion of the software. This coupling between components can cause critical problems in the development cycle, increasing the costs of its evolution and maintenance. This paper presents DemeterWatch, a tool to identify a specific case of coupling violations inside Java projects source code.

Statement of need

One of the pillars of software engineering is the pursuit of continuous improvement of processes and practices, incorporating feedback and lessons throughout the software lifecycle. Proper documentation, efficient team collaboration, and ensuring quality are fundamental elements for success in this type of project. Additionally, the rapid evolution of technology requires software engineering professionals to stay up-to-date and ready to adapt to new trends and challenges in the industry, with software quality assurance being one of the key practices in the field ([Pressman & Maxim, 2014](#)).

During the process of software development, a recurring issue is the coupling created between classes, often due to inefficient modeling, resulting in code with classes and methods that are not very reusable and difficult to maintain during the other stages. A principle aimed at minimizing the problems mentioned above is the Law of Demeter (LoD) or the principle of the least knowledge ([Lieberherr et al., 1988](#)). When a class does not comply with the LoD, other problems may arise. One such issue is the confinement break, which occurs when an object can change the state of another object without its knowledge, by calling methods that improperly expose their properties.

Identifying principle violations in Java source code is challenging. Static analysis ([Bardas & others, 2010](#)), the method employed in this work, cannot pinpoint which methods alter an object's state. Consequently, the analysis is focused on data structures within the Java Collections Framework (JCF) ([Collections Framework Overview — Docs.oracle.com](#)). The JCF is well-documented, and its documentation specifies which methods modify the state of collection objects.

This approach is crucial as it highlights issues that are not readily apparent, including subtle and difficult-to-detect forms ([Bock, 2018](#)). It also provides a means to visualize detected

breaches, facilitating corrections in the source code.

State of field

Some works have already attempted to identify such violations in other software contexts and scopes. One example is Hou et al. (2004), which presented a specification language for use by framework users, enabling them to mechanically check code compliance. This allows framework developers to create constraints on what others must avoid when coding with that framework in a C++ environment. It also demonstrates the creation of a Law of Demeter specification outlining which couplings are acceptable and which should be avoided. Unfortunately, it did not show any ways to detect when an object's state is being changed, which is the focus of this project.

In a second work, Chiba et al. (2010) aimed to detect Law of Demeter (LoD) violations within Java as an Eclipse plugin, but it primarily focused on checking for dependencies between packages.

Unlike the tools cited above, this work contributes to the state of the field by delving into the detection of not only pure LoD violations but also whether these violations change the object's state without its consent, because this leads to a more unpredictable context and a more difficult software to maintain.

Example use case

The complete dataset of experiments is available in the repository to further visualization containing forty randomly selected projects outputs, those projects are present at Qualitas.class corpus (Terra et al., 2013). Table 1 shows how many LoD violations were found in open source projects, some of them well-known projects used by Java developers:

Name	Violations found
Apache Collections	6
Apache Tomcat	20
Quartz	1
JGraphT	6
JMeter	6

Table 1. Example of violations found in real open source projects analyzed by the tool.

In addition to textual output, the tool generates an HTML page with a graphical representation of method call chains in the source code. Nodes that violated LoD principle are highlighted in a distinct color. Each vertex's weight indicates the frequency of method calls. Figure 1 contains an illustrated example of a detected violation.

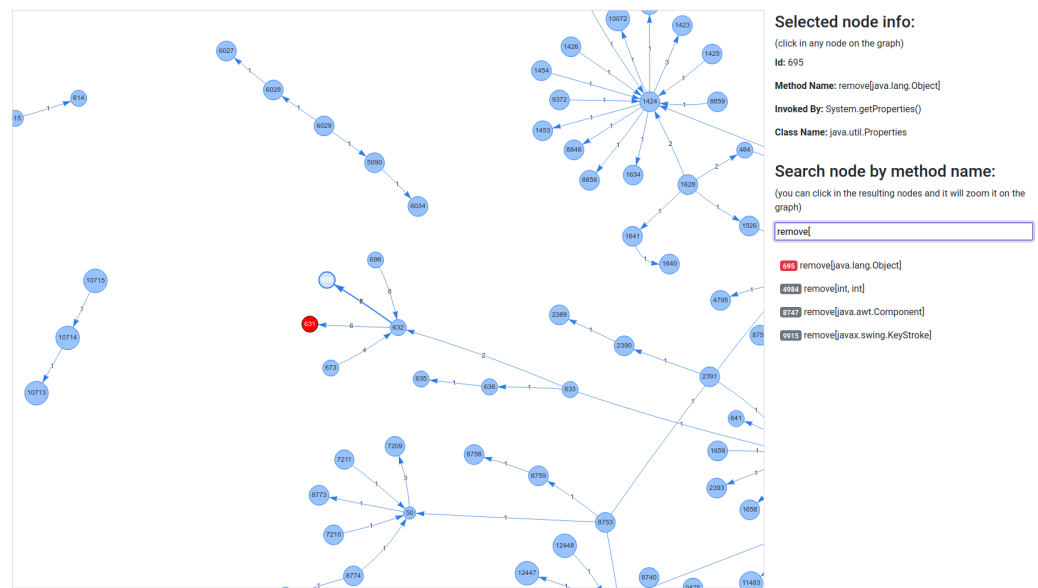


Figure 1. DemeterWatch HTML output page.

Acknowledgements

The main author, Juan Pablo P. de Aquino, would like to express gratitude to the IFPB Institute which made this research project possible to be developed and offered the support needed.

References

- Bardas, A. G., & others. (2010). Static code analysis. *Journal of Information Systems & Operations Management*, 4(2), 99–107.
- Bock, D. (2018). *The paperboy, the wallet, and the Law of Demeter*. <https://www2.ccs.neu.edu/research/demeter/demeter-method/LawOfDemeter/paper-boy/demeter.pdf>
- Chiba, R., Hashiura, H., & Komiya, S. (2010). A tool for detecting detects on class implementation in object oriented program on the basis of the Law of Demeter: Focusing on dependency between packages. *Proceedings of the 10th WSEAS International Conference on Applied Computer Science*, 315–320. ISBN: 9789604742318
- Collections Framework Overview* — docs.oracle.com. <https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html>.
- Hou, D., Hoover, H. J., & Rudnicki, P. (2004). Specifying the Law of Demeter and C++ programming guidelines with FCL. *Source Code Analysis and Manipulation, Fourth IEEE International Workshop on*, 119–127. <https://doi.org/10.1109/scam.2004.22>
- Lieberherr, K., Holland, I., & Riel, A. (1988). Object-oriented programming: An objective sense of style. *SIGPLAN Not.*, 23(11), 323–334. <https://doi.org/10.1145/62084.62113>
- Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12), 1053–1058. <https://doi.org/10.1145/361598.361623>
- Pressman, R. S., & Maxim, B. R. (2014). *Software engineering: A practitioner's approach* (8th ed.). McGraw-Hill Professional. ISBN: 9780078022128
- Terra, R., Miranda, L. F., Valente, M. T., & Bigonha, R. S. (2013). Qualitas.class corpus: A compiled version of the qualitas corpus. *SIGSOFT Softw. Eng. Notes*, 38(5), 1–4.

<https://doi.org/10.1145/2507288.2507314>