On Quantifying the Benefits of Dead Code Removal

Kudrjavets, Gunnar; Rastogi, Ayushi; Thomas, Jeffrey; Nagappan, Nachiappan

Published in:
2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)

DOI:
10.1109/ICSME55016.2022.00076

On Quantifying the Benefits of Dead Code Removal

Gunnar Kudrjavets, Ayushi Rastogi, Jeff Thomas, Nachiappan Nagappan

On Quantifying the Benefits of Dead Code Removal

Gunnar Kudrjavets, Ayushi Rastogi
University of Groningen
9712 CP Groningen, Netherlands
g.kudrjavets@rug.nl, a.rastogi@rug.nl

Jeff Thomas, Nachiappan Nagappan
Meta Platforms, Inc.
Menlo Park, CA 94025, USA
jeffdthomas@fb.com, nnachi@fb.com

***Abstract*—Engineers consider the presence of dead code as an undesirable attribute of the code base. The industry lacks methods to quantify the benefits of deleting dead code efficiently. The current approach utilizes a simplistic metric that uses the lines of code (LOC) deleted as a proxy to estimate the benefit gained. However, not all LOC are equal. The research community can support the industry and propose methods and metrics that can help to (a) determine the priority order for dead code removal, and (b) quantify the benefits of dead code removal. Improved metrics can result in a more objective ranking of dead code deletion efforts when compared to other competing tasks.**

Introduction. Dead code is code that is either needlessly executed or code that a system contains but never runs. Unnecessary code is a subcategory of technical debt and a “bad code smell” [1]. It is also a cause for maintenance-related issues [2]. The presence of dead code is one of the categories in the Common Weakness Enumeration system [3]. In programmer folklore, dead code is an undesirable trait [4].

The first automated attempt to remove dead code happens during the compilation. Compilers use techniques such as interprocedural optimization [5] to remove dead code. Approaches like static analysis can supplement compiler optimizations [6]. Tools can efficiently determine if code is not called and optimize it away. However, *automation cannot identify code that becomes unnecessary because of either changing requirements or obsolete features*. These decisions require human intervention.

Organizations use various human-centric approaches to remove dead code systemically. For example, they establish virtual teams consisting of “janitors” [7] or designate a set of engineers as the “Code Cleanup Crew.” Organizations may also implement dedicated global engineering efforts that are either scheduled or continuous. Online services companies A and B resort to a mix of material and social rewards. In our experience at company A, engineers who delete at least N LOC acquire a dedicated badge on an internal employee profile page, get a limited-edition t-shirt, and become members of the selective “Dead Code Society.”

Problem. Each organization or project has several competing priorities. Organizations can invest engineering effort into activities such as developing new features, fixing existing defects, or improving a product’s performance. Intuitively engineers know the tax dead code imposes on the development process. In *commercial software development*, organizations decide where to invest engineering resources by calculating the estimated return on the investment. *There are no straightforward approaches to estimating the value gained from deleting dead code*. Compared to other engineering activities, the benefits of which are well-known, calculating the estimated return on investment for dead code deletion presents a challenge.

The existing framework of “the bigger the number of deleted LOC, the better” is overly simplistic. This valuation scheme does not correctly quantify the benefits gained from deleting dead code. *Not all lines of code are equal*. The benefit from dead code deletion depends on a variety of contextual factors such as (a) abstraction level (e.g., kernel mode versus user mode), (b) performance cost (e.g., variable initialization versus creating a thread), and (c) exposure and intended usage (e.g., primary attack surface versus a rarely used feature).

Challenges to the research community. The research community can help practitioners by (a) conducting studies on systems that use a systematic approach to delete dead code (e.g., an effort to “prune” code in OpenBSD [8]) to investigate if other system characteristics such as performance, quality, or the perceived cleanliness of code base change as a result, (b) providing means to compose guidance about the order in which to focus on different system components (e.g., based on abstraction level, code coverage, or attack surface), and (c) defining methods to quantify the benefits of deleting dead code that are more precise than the number of LOC deleted.

References

[1] S. Romano, C. Vendome, G. Scanniello, and D. Poshyvanyk, “A multi-study investigation into dead code,” *IEEE Transactions on Software Engineering*, vol. 46, no. 1, pp. 71–99, 2020. [Online]. Available: https://doi.org/10.1109/TSE.2018.2842781

[2] S. Eder, M. Junker, E. Jürgens, B. Hauptmann, R. Vaas, and K.-H. Prommer, “How much does unused code matter for maintenance?” in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE ’12. IEEE Press, 2012, p. 1102–1111. [Online]. Available: https://doi.org/10.1109/ICSE.2012.6227109

[3] “CWE - CWE-561: Dead Code (4.6).” [Online]. Available: https://cwe.mitre.org/data/definitions/561.html

[4] G. V. Neville-Neil, “Removing Kode,” *Commun. ACM*, vol. 63, no. 12, p. 29, nov 2020. [Online]. Available: https://doi.org/10.1145/3430378

[5] S. K. Debray, W. Evans, R. Muth, and B. De Sutter, “Compiler techniques for code compaction,” *ACM Trans. Program. Lang. Syst.*, vol. 22, no. 2, p. 378–415, Mar 2000. [Online]. Available: https://doi.org/10.1145/349214.349233

[6] R. Haas, R. Niedermayr, T. Roehm, and S. Apel, “Is static analysis able to identify unnecessary source code?” *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 1, jan 2020. [Online]. Available: https://doi.org/10.1145/3368267

[7] J. D. Morgenthaler, M. Gridnev, R. Sauciuc, and S. Bhansali, “Searching for build debt: Experiences managing technical debt at Google,” in *Proceedings of the Third International Workshop on Managing Technical Debt*, ser. MTD ’12. IEEE Press, 2012, p. 1–6.

[8] T. Unangst, “Pruning and Polishing: Keeping OpenBSD Modern,” in *Proceedings of AsiaBSDCon 2015*. Tokyo, Japan: Tokyo University of Science, Mar. 2015. [Online]. Available: https://www.openbsd.org/papers/pruning.html