What Warnings Do Engineers Really Fix? The Compiler That Cried Wolf

Gunnar Kudrjavets
University of Groningen
9712 CP Groningen, Netherlands
g.kudrjavets@rug.nl

Aditya Kumar
Snap, Inc.
Santa Monica, CA 90405, USA
adityak@snap.com

Ayushi Rastogi
University of Groningen
9712 CP Groningen, Netherlands
a.rastogi@rug.nl

***Abstract*—Build logs from a variety of Continuous Integration (CI) systems contain temporal data about the presence and distribution of compiler warnings. Results from the analysis and mining of that data will indicate what warnings engineers find useful and fix, or continuously ignore. The findings will include resolution times and resolution types for different warning categories. That data will help compiler developers adjust the warning levels according to the ground truth, clarify the diagnostic messages, and improve the non-actionable warnings. The empirical findings will also help engineers to decide what warnings are worth fixing and which ones are not.**

I. Background and motivation

Compiler warnings that indicate *potential* problems in the code are a byproduct of a daily "edit-compile-debug" development cycle. If warnings are present, an engineer must decide whether to fix, ignore, or suppress the warnings. Industry practices for commercial and open-source projects range from a zero-tolerance policy towards warnings [1], [2], [3] to stating that fixing warnings introduces more defects [4].

Ignoring critical compiler warnings can cause defects to propagate into a production environment. Those defects, in turn, can trigger issues such as reduced availability of services, memory corruption, and zero-day vulnerabilities. We need empirical data to develop sound, evidence-based policies to determine if and in what order to address compiler warnings.

Google's experience with applying static analysis at a large scale indicates that *it is hard to motivate developers to fix potential issues without a clear incentive* [5]. In our industry experience, we make concurring observations regarding prioritizing fixing compiler warnings. Lack of evidence that can provide the incentive makes it challenging to determine the priority of this work. Outside of the warning level ranking provided by the compiler, we are not aware of any other prioritization techniques to determine what warnings should be fixed first. Our industry experience indicates that *some warnings are more beneficial than others*. Benefit in this context means that warnings detect serious defects, have less false positives, and engineers trust and understand the diagnostic reports [6], [7]. We anecdotally observe that *the order in which engineers fix the warnings does not always match the severity level a compiler assigns to them*. To determine the types of warnings that engineers either continuously ignore, fix immediately, or delay fixing, we propose that researchers mine the existing build logs and investigate this topic further.

II. Availability of data

CI is a practice of continuously integrating code changes into a shared code base [8]. The CI systems are widely used in industry by companies such as Google [9] and Meta [10]. The adoption of CI systems for open-source software is growing as well [11]. Several popular open-source projects such as FreeBSD [12] or Mozilla [13] have embraced the CI systems as the primary code delivery and validation vehicle.

The CI process results in a large number of build artifacts. In 2017, Google executed approximately 800,000 builds on a daily basis [9]. The build logs contain compiler output that includes the list of warnings that were generated during the compilation process. Typically, each CI build tracks also what new code changes are included in that build. Popular dynamic and static analysis frameworks such as CodeChecker enable inspecting the differences between two builds [14]. In our industry experience, the resulting build logs persist anywhere from weeks to months. As a result, a variety of data from both industry and open-source is available for analysis.

III. Future research topics

The repository of CI logs can help researchers find answers to several questions such as:

1) What warnings have the longest and shortest lifespan? Does the order of fixing those warnings correspond with compiler's classification scheme and suggestions?
2) Are warnings from some compilers fixed faster than others (e.g., Clang versus GCC)? What about different programming languages or static analysis tools?
3) Is there a relationship between various characteristics such as abstraction level, amount of code churn, programming language, or seniority of engineers and how fast the warnings are fixed?
4) What is the temporal change direction in the ratio of warnings to source lines of code? Does the ratio depend on different project characteristics?

Answering these questions will help engineers prioritize fixing compiler warnings, suppress the ineffective warnings, and provide feedback to compiler developers. Combing the data mined from CI logs with the information from the defect tracking database and source control system will provide even stronger evidence to substantiate any claims.

References

[1] G. J. Holzmann, "The power of 10: rules for developing safety-critical code," *Computer*, vol. 39, no. 6, pp. 95–97, Jun. 2006. [Online]. Available: https://doi.org/10.1109/MC.2006.212

[2] N. Nethercote. (2017, Jul.) How we made compiler warnings fatal in Firefox. Mozilla Foundation. [Online]. Available: https://blog.mozilla.org/nnethercote/2017/07/05/how-we-made-compiler-warnings-fatal-in-firefox/

[3] The Apache Software Foundation. (2021, Jun.) Writing warning-free code—Apache OpenOffice Wiki. [Online]. Available: https://wiki.openoffice.org/wiki/Writing_warning-free_code

[4] SQLite. (2022, May) How SQLite Is Tested. SQLite Consortium. [Online]. Available: https://www.sqlite.org/testing.html#staticanalysis

[5] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, "Lessons from building static analysis tools at Google," *Commun. ACM*, vol. 61, no. 4, pp. 58–66, mar 2018. [Online]. Available: https://doi.org/10.1145/3188720

[6] T. Barik, J. Smith, K. Lubick, E. Holmes, J. Feng, E. Murphy-Hill, and C. Parnin, "Do Developers Read Compiler Error Messages?" in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. IEEE Press, 2017, pp. 575–585. [Online]. Available: https://doi.org/10.1109/ICSE.2017.59

[7] B. A. Becker, P. Denny, R. Pettit, D. Bouchard, D. J. Bouvier, B. Harrington, A. Kamil, A. Karkare, C. McDonald, P.-M. Osera, J. L. Pearce, and J. Prather, "Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research," in *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 177–210. [Online]. Available: https://doi.org/10.1145/3344429.3372508

[8] M. Fowler. (2006, May) Continuous Integration. [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html

[9] A. Memon, Z. Gao, B. Nguyen, S. Dhanda, E. Nickell, R. Siemborski, and J. Micco, "Taming Google-scale continuous testing," in *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, ser. ICSE-SEIP '17. IEEE Press, 2017, pp. 233–242. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2017.16

[10] D. Distefano, M. FÃ€hndrich, F. Logozzo, and P. W. O'Hearn, "Scaling static analyses at Facebook," *Communications of the ACM*, vol. 62, no. 8, pp. 62–70, Jul. 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3338112

[11] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2016. New York, NY, USA: Association for Computing Machinery, 2016, pp. 426–437. [Online]. Available: https://doi.org/10.1145/2970276.2970358

[12] L.-W. Hsu, "Continuous Integration of The FreeBSD Project," in *Proceedings of AsiaBSDCon 2017*. Tokyo, Japan: Tokyo University of Science, Mar. 2017. [Online]. Available: https://papers.freebsd.org/2017/asiabsdcon/lwhsu-Continuous_Integration_of_the_FreeBSD_Project.files/lwhsu-Continuous_Integration_of_the_FreeBSD_Project-paper.pdf

[13] J. Lampel, S. Just, S. Apel, and A. Zeller, "When life gives you oranges: detecting and diagnosing intermittent job failures at Mozilla," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1381–1392. [Online]. Available: https://doi.org/10.1145/3468264.3473931

[14] G. Márton and D. Krupp. (2020, Jun.) Tool Talk: CodeChecker (SOAP 2020—9th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis (SOAP) 2020)—PLDI 2020. [Online]. Available: https://pldi20.sigplan.org/details/SOAP-2020-papers/13/Tool-Talk-CodeChecker