Keynote Talk: On Estimating Incorrectness in the Absence of Evidence

Marcel Böhme
marcel.boehme@acm.org
MPI for Security and Privacy
Germany

Abstract

In this keynote, we introduce a statistical framework for automated software testing, including fuzzing, that allows us to estimate, extrapolate, and explain various interesting properties of a testing campaign. We show how to quantify our confidence in the correctness of a program after a long-running testing campaign that has found no bugs. We explain how Shannon's entropy is a non-parametric measure of testing efficiency and why the cost of bug finding appears to be exponential even if testing is embarrassingly parallel. We extend the approach to statistical program analysis and make progress on the statistical problem itself. Finally, we talk about our most recent work on estimating incorrectness (of an LLM-generated program) in the absence of an oracle.

CCS Concepts

• **Software and its engineering → Software testing and debugging**.

Keywords

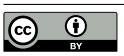
Empirical Program Analysis, Oracle Problem, Residual Risk

ACM Reference Format:
Marcel Böhme. 2026. Keynote Talk: On Estimating Incorrectness in the Absence of Evidence. In *Innovations in Software Engineering Conference (ISEC 2026), February 19–21, 2026, Jaipur, India.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3796563.3796565

1 Estimating Error in the Absence of Evidence

> Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.
>
> *Edsger W. Dijkstra; ACM Turing Lecture 1972*

The utility of automated software testing as a bug finding technique is entirely obvious; but what if a long-running testing campaign finds no bugs? In practice, this is (thankfully!) the much more frequent case. The question that naturally arises is: What does the absence of evidence in an ongoing testing campaign say about the absence of bugs? Intuitively, we will have more confidence in a campaign that has run for one month than in one that has run for one minute. Can we objectively and systematically quantify this confidence in the presence of uncertainty?



ISEC 2026, Jaipur, India

ACM ISBN 979-8-4007-2503-6/2026/02
https://doi.org/10.1145/3796563.3796565

In this keynote, we introduce a statistical framework for automatic software testing [1] which allows us to explain various, interesting phenomena of automatic software testing that practitioners have previously observed but that have never been empirically studied or formally explained. We explain why whitebox fuzzing is conceptually the most effective testing technique, and why, nevertheless, black- and greybox fuzzing often outperform whitebox fuzzing, in practice [4]. We discuss why the cost of bug finding appears to be exponential even though testing is embarrassingly parallel [2]. We formally establish Shannon's entropy, a well-known measure of information and diversity [7], as non-parametric measure of testing efficiency [3] and discuss how our entropy-based power schedule ended up powering large fuzzing infrastructures at Google and Microsoft where it helps to continuously sweep thousands of important software systems for critical security flaws.

Meanwhile, we show how our statistical framework has evolved, from its initial goal of quantifying the confidence which a testing campaign inspires that has found no bugs, to a framework for the statistical analysis of a wide variety of binary[1] or quantitative[2] program properties, particularly in the absence of evidence [5]. For instance, while we can, in general, not *decide* whether even a single, specific program state is reachable, we can efficiently *estimate* the total number of reachable states (or coverage elements) [6].

2 Estimating Error in the Absence of an Oracle

So far, we have assumed that there exists an automatic oracle that decides for any execution whether or not a bug is revealed. A typical example of automatic oracles in fuzzing is the crash oracle which may indicate a security flaw. However, in automatic code generation, where an LLM or an agentic system translates a programming task, expressed in natural language, into a program that implements the task, such an oracle does not practically exist.

In the absence of such an oracle, can we estimate the likelihood that the generated program is correct? We explore this question in the second part of the keynote with reference to a recent paper that appears at AAAI'26 [8]. The key idea is to consider the program, that is generated for a programming task, as a random variable. This allows us to introduce a probabilistic notion of correctness with respect to an unknown, deterministic ground-truth implementation for that task. We approximate the expected incorrectness using a quantity, we define as *incoherence*, i.e., the probability that two programs randomly generated for the same task compute different output for some input. We offer PAC-style bounds to establish sampling efficiency and propose to leverage the statistical framework introduced in the first part to estimate incoherence in practice.

[1] An execution does or does not exhibit a specific behavior.
[2] An example of a quantitative property is the time an execution takes.

Acknowledgments

I would like to express my gratitude to the Chairs of the ISEC Program Committee Rupak Majumdar and Subhajit Roy for their kind invitation, as well as the General Chairs Anubha Jain and Vikas Bajpai and their organizing committee for their wonderful support and organization. I thank ACM India for the kind sponsorship.

References

[1] Marcel Böhme. 2018. STADS: Software Testing as Species Discovery. *ACM Transactions on Software Engineering and Methodology* 27, 2, Article 7 (June 2018), 52 pages. doi:10.1145/3210309

[2] Marcel Böhme and Brandon Falk. 2020. Fuzzing: on the exponential cost of vulnerability discovery. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 713–724. doi:10.1145/3368089.3409729

[3] Marcel Böhme, Valentin J. M. Manès, and Sang Kil Cha. 2023. Boosting Fuzzer Efficiency: An Information Theoretic Perspective. *Commun. ACM* 66, 11 (Oct. 2023), 89–97. doi:10.1145/3611019

[4] Marcel Böhme and Soumya Paul. 2016. A Probabilistic Analysis of the Efficiency of Automated Software Testing. *IEEE Transactions on Software Engineering* 42, 4 (2016), 345–360. doi:10.1109/TSE.2015.2487274

[5] Seongmin Lee and Marcel Böhme. 2023. Statistical Reachability Analysis. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (San Francisco, CA, USA) *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 326–337. doi:10.1145/3611643.3616268

[6] Danushka Liyanage, Marcel Böhme, Chakkrit Tantithamthavorn, and Stephan Lipp. 2023. Reachable Coverage: Estimating Saturation in Fuzzing. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 371–383. doi:10.1109/ICSE48619.2023.00042

[7] Hoang Lam Nguyen and Lars Grunske. 2022. BEDIVFUZZ: Integrating Behavioral Diversity into Generator-based Fuzzing. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 249–261. doi:10.1145/3510003.3510182

[8] Thomas Valentin, Ardi Madadi, Gaetano Sapia, and Marcel Böhme. 2026. Incoherence as Oracle-less Measure of Error in LLM-Based Code Generation. In *Proceedings of the 40th Annual AAAI Conference on Artificial Intelligence (AAAI'26)*. 14 pages.