

Assurance in Software Testing: A Roadmap

Marcel Böhme
Monash University
marcel.boehme@acm.org

Abstract—As researchers, we already understand how to make testing more effective and efficient at finding bugs. However, as fuzzing (i.e., automated testing) becomes more widely adopted in practice, practitioners are asking: Which assurances does a fuzzing campaign provide that exposes no bugs? When is it safe to stop the fuzzer with a reasonable residual risk? How much longer should the fuzzer be run to achieve sufficient coverage?

It is time for us to move beyond the innovation of increasingly sophisticated testing techniques, to build a body of knowledge around the explication and quantification of the testing process, and to develop sound methodologies to estimate and extrapolate these quantities with measurable accuracy. In our vision of the future practitioners leverage a rich statistical toolset to assess residual risk, to obtain statistical guarantees, and to analyze the cost-benefit trade-off for ongoing fuzzing campaigns. We propose a general framework as a first starting point to tackle this fundamental challenge and discuss a large number of concrete opportunities for future research.

I. INTRODUCTION

Cognitive psychology tells us that the unaided human mind is vulnerable to many fallacies and illusions because of its reliance on its memory for vivid anecdotes rather than systematic statistics.

— Prof. Steven Pinker (Dep. of Psychology at Harvard)

A. A Vivid Anecdote

Last year, we attended a meeting with several representatives of a large company that provides security assessment services for governments and industries worldwide. In preparation for this meeting, we learned about their product portfolio and found that their main product, a protocol-based blackbox fuzzer,¹ can be used for *security certification of medical devices* (IEC 62443-4-2).² While the fuzzer is part of a larger certification process,³ it is primarily the fuzzer’s task to identify vulnerabilities that could be exploited remotely over the network.

Now, medical devices are safety-critical systems and undetected vulnerabilities can be life threatening. For instance, Halperin et al. [2] describe several attacks on an implantable cardioverter defibrillator to control when electrical shocks are administered to the patient’s heart. Hence, subjecting medical devices to rigorous cyber security assessment is a powerful mitigator of cyber risks. This inspires *trust* in the certificate.

¹We shall use the terms *fuzzing* and *automated testing* interchangeably.

²The assessment scheme for IEC 62443, the worldwide standard for security of Industrial Control Systems, is operated by the ISA Security Compliance Institute and offered within the Embedded Device Security Assurance (EDSA) product which ascertains compliance with IEC 62443-4-2 [1].

³The ISASecure EDSA certification also requires that the organization follows a robust, secure software development process and that the product has properly implemented the security-related functional requirements.

A violation of this trust (e.g., if an attacker exploited an undetected vulnerability in a medical device that is *certified*) would be disastrous. Thus, the company’s reputation and the certificate’s credibility depend, at least in part, on the *assurances* which the fuzzer provides.

We walked into that meeting wondering about this question. Which assurances are derived for the certificate from applying the company’s fuzzer? To paraphrase Dijkstra, fuzzing can be used only to show the presence of vulnerabilities, not their absence. How do they effectively assess the residual risk of a fuzzing campaign that finds no vulnerabilities? How do they arrive at the decision to stop the fuzzer and to proceed with the certification? It turns out that the certification scheme does not specify how much fuzzing is sufficient for certification. Neither does it specify a concrete value for the *allowable* residual risk that an undiscovered vulnerability still exists. In practice, *the decision is with the individual security researcher*. Even if a concrete threshold value was specified, there is no statistical framework available that would allow the researcher to quantify the residual risk for an ongoing campaign [3]–[5].

In the end, we were told, the decision is mostly based on *experience*. Clearly, a long-running fuzzing campaign provides much stronger assurances than a shorter one; meaning, the residual risk decreases as the length of the campaign increases. Hence, intuitively there is a particular point in time when it is both economical and safe to abort a fuzzing campaign that has found no vulnerabilities.

B. Call for Systematic Statistics

In this paper, we argue that we ought to do better than relying on an individual’s experience. The security researcher should be able to systematically assess and quantify the inherent uncertainty. The certification authority should be able to provide concrete guidance in the form of measurable threshold values. This offers an opportunity for us as software engineering *researchers* to develop a rich statistical toolset that will enable software engineering *practitioners* to assess and quantify the automated testing process.

Senior members of our community have previously called for a general statistical framework. In 2000, Harrold [3] established the “development of techniques and tools for use in *estimating, predicting, and performing testing*” as a key research objective for future software testing research. Seven years later, Bertolino [4] corroborated that “we will need to make the process of testing more effective, *predictable* and *effortless*”. Yet today, Whalen [5] observes “there is *no sound basis to extrapolate* from tested to untested cases”.⁴

⁴In the quotes, the emphasis in *italic letters* is mine.

Over the last couple of years, fuzzing has become widely adopted in practice, including by Google [6], Microsoft [7], Adobe [8], Mozilla [9], and Facebook [10]. Arguably, we already know how to *control, bias, and optimize* the fuzzing process so as to make it more efficient [11], [12]. Search-Based Software Testing (SBST) has become a mature field of research [13]. Practitioners utilize a large collection of extremely efficient fuzzing strategies.

Going forward, there needs to be a similarly rich body of work that facilitates a deeper understanding how to *quantify, measure, and assess* the fuzzing process so as to make it more explicable. Only then, practitioners will be able to answer very practical questions, such as

- 1) **Residual risk.** Supposing no vulnerability has been found after generating n test inputs, what is the probability to generate a new test input that does expose a vulnerability? More generally, *how much has been learned about the program's behavior, and how much more remains to be learned?*
- 2) **Cost-benefit Trade-off.** Supposing no vulnerability has been found after generating n test inputs, how does the residual risk decrease if time is spent generating m more test inputs? More generally, *how much more could be learned about the program's behavior within an additional time budget?*
- 3) **Testability.** How “difficult” is it for an arbitrary fuzzer to discover vulnerabilities in the program? More generally, *what is the average amount of information that the program reveals about its behaviors per generated test input?*
- 4) **Effectiveness.** How effective is the fuzzer w.r.t. discovering vulnerabilities? More generally, *what is the asymptotic number of (discrete) program behaviors that the fuzzer is able to expose?*

So then, why have previous calls of eminent researchers for extrapolation in software testing gone unheeded? To most of us, it seems counter-intuitive that one can soundly extrapolate from tested to untested program behaviors. Rushby (1992) [14] explains by analogy: A physical system exhibits behavior that is *continuous*, which allows engineers to test a few critical points of a bridge to make predictions about the safety of the whole bridge. In contrast, a software system exhibits behavior that is *discretized* and *discontinuous*. Hence, “tests provide information on only the state sequences actually examined; without continuity there is little reason to suppose the behavior of untested sequences will be close to tested ones, and therefore little justification for extrapolating from tested cases to untested ones” [14]. However, in the last 16 years statistics has made huge strides, providing statistical tools that make *no assumptions about continuity* (here, of program behaviors).

Moreover, many of us with a formal methods background are swayed by counter-examples. It is trivial to construct a small program with an `if`-condition where the `else`-branch is evaluated with an infinitesimal probability. How can one ever expect to extrapolate in a sound manner? The answer is two-pronged: (i) It is possible to make *no assumptions* about the probability of some behavior to be observed or how many behaviors there are. (ii) Empirically, the typical program does not resemble such counter-examples. Indeed, an empirical evaluation of the performance of several biostatistical estimators for software testing showed tremendous promise [15].

II. A GENERAL STATISTICAL FRAMEWORK FOR SOFTWARE TESTING: CHALLENGES AND OPORTUNITIES

Estimation and extrapolation are classical problems in statistics. We can understand testing essentially as a sampling of program behaviors. As more tests are executed, we learn more about the program's behaviors. Our confidence in the program's correctness increases. To quantify this confidence (and generally the uncertainty about any concrete statements made, given only limited data) is a fundamental building block of statistics. Hence, it stands to reason to construct a *general statistical framework* for software testing.

An interesting initial attempt in this direction is the STADS framework [15]. Böhme borrows several innovative biostatistical methodologies from ecology to answer those practitioners' questions (on the left) about residual risk, cost-benefit, testability, and effectiveness. The ecologic analogy is very powerful. It addresses Rushby's concerns [14] about discontinuity and provides direct access to over thirty years of research in ecological biostatistics. However, while STADS is an excellent starting point, the author introduces only a limited number of estimators and leaves many questions unanswered. Major hurdles are still ahead of us, providing abundant opportunities for future research.

It is also the purpose of this article to highlight these peculiarities, to discuss concrete opportunities for future work, and to elucidate the underlying assumptions, their impact on the statistical guarantees, and how they can be addressed. These are the construction sites on our path to a fundamental understanding of the testing process.

A. Software Testing as Species Discovery

Ecology is concerned with species discovery. For instance, to study the species diversity of arthropods in a tropical rain forest, ecologists would first sample a large number of individuals from that forest and determine their species. The species in the sample are said to be *discovered*. Exhaustive sampling is often prohibitive⁵ and the discovered species represent only a proportion of all species. Hence, the fundamental challenge is to *extrapolate from the species observed in the sample*. Biostatisticians spent the last three decades [17] developing a framework that addresses this extrapolation challenge in ecology.

Böhme's [15] key observation was that testing is about *discovery*, as well. For instance, let us call each branch that is covered by an input, a *species of that input*. To maximize branch coverage, a fuzzer would first sample a large number of test inputs from that program's input space and determine their species. The species in the sample are said to be discovered. Again, exhaustive sampling is prohibitive and the discovered species (here, the covered branches) represent only a proportion of all species (here, all feasible branches). Again, the fundamental challenge is to *extrapolate from the species observed in the sample of generated test inputs*.

⁵For instance, it took 102 ecology researchers 66 person-years to sample 129,494 arthropod individuals representing 6144 species from 0.48 ha of tropical rain forest [16].

The species for an input can be defined in several ways, depending on the dynamic program properties of interest. We could say a new species is discovered when the fuzzer generates an input that

- exercises a statement, branch, path, or any other program element not previously exercised (code coverage),
- kills a mutant not previously killed (mutation-adequacy),
- exposes a previously unexposed assertion violation, program crash, memory error, non-functional error, race condition, etc. (bug finding),
- discovers a previously undiscovered sensitive information flow [18],
- or any other discrete property of the program’s behavior,
- or any combination of the above.

Within the ecologic analogy, a *test input* is the individual or sampling unit, and the observed program behavior is the input’s species. An input can belong to multiple *species*. For instance, each statement that an input executes may be considered a species of that input. A *fuzzer* samples test inputs and discovers species. A *dynamic analysis* identifies an input’s species. For instance, the `gcc` tool identifies the statements exercised by an input.

The perspective of Software Testing and Analysis as Discovery of Species (STADS) provides direct access to 30+ years of research in ecological biostatistics [17]. In the following, we present examples of biostatistical estimators and their utility in automated testing.⁶

Residual risk can be assessed using the probability to discover a new species.⁷ The *discovery probability* is the probability that the next generated test input leads to the discovery of a previously unseen species. The discovery probability $U(n)$ can be estimated accurately and efficiently for arbitrary species abundance distributions [19]–[22] using Good-Turing [23]

$$\hat{U}(n) = \frac{f_1}{n} \quad (1)$$

where f_1 is the number of (singleton) species having been observed exactly once in the campaign.⁸ Now, a test input can belong to some species and *not* expose an error, or belong to the same species and expose an error. For the purpose of discussion let the latter be a species on its own. If no error has been exposed throughout the campaign, the Good-Turing estimator gives an upper bound on the probability to generate a test input that exposes an error.

⁶Due to the lack of space, the reader is referred to Böhme [15] for a formal introduction of the STADS framework and the statistical models underpinning the framework.

⁷In ecology, discovery probability is called sample coverage and gives the proportion of individuals in the population belonging to a species represented in the sample.

⁸Intuitively, even after n test inputs have been generated there are still f_1 species that have been observed exactly once; clearly, we can expect it takes at least as many test inputs to observe one of f_0 undiscovered species that have not been observed, at all.

Cost-benefit trade-offs can be assessed using statistical extrapolation. Given $S(n)$ species have been discovered after n test inputs have been generated, we can estimate the number of species $S(n + m^*)$ —that we can expect to discover if m^* more test inputs are generated—using the following estimator [17], [24]

$$\hat{S}(n + m^*) = S(n) + \hat{f}_0 \left[1 - \left(1 - \frac{f_1}{n\hat{f}_0 + f_1} \right)^{m^*} \right] \quad (2)$$

where f_1 is the number of (singleton) species having been observed exactly once in the campaign, and $\hat{f}_0 = \hat{S} - S(n)$ is an estimate of the number of (undiscovered) species. If unknown, the asymptotic total number of species S can be estimated as follows [25], [26]

$$\hat{S} = \begin{cases} S(n) + f_1^2/(2f_2) & \text{if } f_2 > 0 \\ S(n) + f_1(f_1 - 1)/2 & \text{otherwise} \end{cases} \quad (3)$$

where f_2 is the number of (doubleton) species that have been observed exactly twice throughout the campaign. Given a discovery probability $U(n)$ after n test inputs have been generated, we can extrapolate the residual risk—in the case that m^* more test inputs were generated—as follows [17], [27]

$$\hat{U}(n + m^*) = \frac{f_1}{n} \left(\frac{n\hat{f}_0}{n\hat{f}_0 + f_1} \right)^{m^*+1} \quad (4)$$

Effectiveness. If the total number of species S is unknown, it can be estimated as given in Equation (3). For instance, if one species corresponds to one mutant in mutation testing, then the \hat{S} estimates the asymptotic number of mutants that *can* be killed. Some mutants are stubborn while others cannot be killed at all [28]. $\hat{G}(n) = S(n)/\hat{S}$ gives an estimate of the *feasible* mutation coverage.

Generality. The STADS framework [15] is *general* in the sense that it does not depend on a specific programming language, or execution environment, testing objective, or test generation technique. For instance, STADS facilitates the extrapolation of statement coverage, mutation adequacy, or the number of bugs exposed for Java, C, and Python programs on Linux, Windows, and MacOS using CSmith [29], Randoop [30], or AFL [31]–[34].

B. Flaky Tests and Non-Deterministic or Stateful Programs

A test is *flaky* if executing it twice on the same program may yield different outcomes. A flaky test may pass nine out of ten times but then fail for no obvious reason. Such a failure may or may not be spurious. Recently, Harman and O’Hearn [10], reflecting on their experience at Facebook, established the presence of flaky tests as a key challenge in software testing research. In fact, researchers should assume that All Tests Are Flaky (ATAF). Statistics is well-equipped to deal with Harman and O’Hearn’s ATAF-istic world.

We say that a test is flaky due to *non-determinism* if the probability of a certain outcome does not change substantially during testing. More specifically, the random outcomes of a flaky test should originate from the same probability distributions, and be mutually independent. In statistics, this is called an independent and identically distributed (*IID*) random variable. Examples are concurrent programs where a particular thread schedule determines the test outcome, or probabilistic programs where the specified branching probabilities determine the test outcome.

We say that a test is flaky due to *statefulness* if the outcomes are not *IID*. The outcomes of a flaky test are either mutually dependent (i.e., induce state changes) or come from different distributions (e.g., due to externally induced state changes). Regardless, there is some underlying state in the program or in its environment that changes in-between test executions. Examples are interactive programs with user-interfaces, such as Android apps or web apps. Executing the same event sequence twice may produce different outputs. For instance, entering a sequence on a calculator program once ('1+1') or twice ('1+1+1') will give different results.

Statistics provides a large set of methodologies for estimation and extrapolation in the presence of tests that are flaky due to non-determinism. In applied statistics, it is a common (though often an unrealistic) assumption that a random variable is *IID* as it drastically simplifies the underlying statistical theory. To determine whether a flaky test is due to non-determinism (i.e., whether its outcomes are *IID*), we suggest the *turning point test*. Existing statistical methodologies should be identified and studied empirically. For instance, to study the performance of an extrapolation methodology in the presence of flaky tests, we can simply compare the predicted value to the actual (future) value in several experiment repetitions.

In order to “unflake” tests that are flaky due to statefulness, we suggest to execute each test case on the *exact same* state. The advent of advanced virtualization and containerization technology allows to capture, control, and restore the entire state of the program and all of its environment, including the entire operating system and the (virtual) machine it is running upon. Clearly, this removes the need to handle tests that are flaky due to statefulness within our statistical framework.

C. Extremely Rare and Rather Extreme Program Behaviors

In software testing, we are often most interested in program behaviors that are both extreme and rarely observable, such as a program crash or a missed deadline in a real-time system. In applied statistics, *rare event analysis* [35] and *extreme value theory* [36] have become substantial fields of research with important applications, e.g., in economics, meteorology, actuarial science, physics, and ecology. For instance, Glasserman et al. [37] propose an adaptive sampling method for the efficient estimation of the probability of a rare event occurring. Software engineering researches should study such highly innovative sampling methods, tailored for the discovery of rare events, to develop more efficient software testing techniques, or to

estimate and extrapolate the residual risk that a vulnerability exists if none has been found.

In software testing, interesting behaviors can be clustered in several very narrow regions of the program’s input space. For instance, some file formats start with a “magic number” [38]; only if that magic number is correct will interesting program behaviors be exercised. In ecology, a large number of *endemic species* may be clustered on remote islands. To facilitate estimation in the presence of endemic species, ecological biostatisticians leverage *sample coverage-based estimators* [39], [40] or more generally *Good-Turing theory* [41]. In future, existing biostatistical estimators should be evaluated empirically and specifically tailored to the context of software testing. There are particularly well-suited adaptive cluster sampling methods for efficient estimation in the presence of a large number of very rare and endemic species [42], [43].

D. Uncertainty about Correctness and the Oracle Problem

The correctness of any statistical guarantee depends on the *soundness of the dynamic program analysis* which distinguishes correct from incorrect program behavior for a given input. More specifically, we cannot assume that *all* inputs that expose a vulnerability are recognized as such [44]. Without a vulnerability-specific dynamic analysis, the fuzzer is unable to automatically *recognize*, e.g., a privilege escalation attack, even if it is actually triggered by a crafted test input (i.e., an exploit). In practice, the statistical guarantees hold only with respect to the kinds of vulnerabilities that *can* be recognized (e.g., a buffer overflow by ASAN [45]). It is interesting to note that a similar challenge exists in software verification where a program can be proven correct only modulo the specification.

Elbaum and Rosenblum [46] discuss the problem of *uncertainty* in software testing. When Amazon recommends to buy Sartre’s “Being and Nothingness”, Spotify recommends to listen to Bach’s “Goldberg variations”, or the GPS is slightly off by a few meters, we can establish their correctness only to some degree of accuracy and with some certainty. Statistics is well-suited for assessing such uncertainties and inaccuracies. This allows to develop techniques that account for the possibility that the oracle (rather than the program) could be incorrect (i.e., cannot be defined with absolute accuracy).

E. Vulnerabilities Outside the Fuzzer’s Search Space

Not all detectable vulnerabilities in the program may be within the fuzzer’s search space. More specifically, even if all inputs that expose a vulnerability are recognized as such, we cannot assume that the fuzzer can generate them. For instance, suppose a vulnerability in an Android app is exposed only via system-level events (e.g., a low battery-level); if the fuzzer cannot generate system-level events, this vulnerability is clearly out of scope for the fuzzer. As such, an estimate of the residual risk quantifies how likely it is that *this* Android fuzzer (or an attacker with a similarly powerful Android fuzzer) generates an event sequence that exposes an undiscovered (but discoverable) vulnerability.

In practice, any statistical guarantees are correct only w.r.t. the detectable vulnerabilities that are within the fuzzer’s search space. In order to provide more general statistical guarantees, we should investigate (i) a sound extrapolation from the fuzzer’s search space to the entire input domain [47] and (ii) a sound integration of several individual estimates—from fuzzers which cover disjoint domains in the program’s input space—into a single, global estimate.

Moreover, we argue that a fuzzer’s effectiveness is defined by the *asymptotic* number of vulnerabilities the fuzzer is able to discover. We should leverage statistical methodologies [25] to extrapolate from the number of vulnerabilities seen throughout a (non-exhaustive) fuzzing campaign to the asymptotic total number of vulnerabilities, in order to derive a *sound estimate of fuzzer effectiveness*. For a sound comparison of two different fuzzers, we should study how to efficiently quantify the overlap of their search spaces.

F. Challenges of Adaptive Bias in Automated Test Generation

Most search-based software testing (SBST) techniques introduce an adaptive bias that needs to be corrected in the statistical analysis. As discussed earlier, the statistical analysis is substantially simplified if we can assume that the outcome of every test the fuzzer generates is *IID*. This is an assumption indeed holds for all blackbox fuzzers that do not adapt based on feedback from the program. However, this assumption does not hold for SBST techniques, where the probability to discover a vulnerability is supposed to increase with the length of the fuzzing campaign. The fitness of previously generated test inputs will adaptively drive the fitness (or quality) of test inputs generated later in the fuzzing campaign.

In ecology, there exist several strategies to correct such adaptive bias a posteriori. Like in software testing, ecologists are interested in boosting the efficiency of species discovery. For instance, the adaptive bias in *adaptive cluster sampling* [48] is corrected using the *Horvitz-Thompson estimator* [49]. For search-based testing, similar bias correction strategies should be developed and empirically evaluated. Unlike in ecology, we can measure the adaptive bias *during* the campaign itself—by comparing predictions to the actual values. This may allow us to dynamically adjust for the bias.

An empirical investigation of the correlation of estimates across various fuzzers for same-length campaigns would bring insights about the estimates’ generality. The program’s source code and program binary provide an additional source of information that can be used to improve estimator performance. In future, the dependence of estimator bias and precision on the discovery probability [23] can be investigated to develop better *bias-correction* mechanisms.

III. CONCLUSION

Coming back to our earlier example: We *can* do better than relying on an individual’s experience when assessing the assurances of automated testing. The *security assessor* can use Good-Turing [23] to estimate (as residual risk) the probability that an attacker—with similar or less resources

and a fuzzer with similar or less efficiency—discovers a vulnerability that the assessor did not discover. The assessor can leverage extrapolation methodologies [17] to predict the reduction of residual risk if there was time to generate m more test inputs. The *certification company* can provide concrete guidance by setting different threshold values for different levels of allowed residual risk. The *security certificate* can provide statistical guarantees and clearly state the conditions under which they hold.

Yet, there are many challenges still ahead of us. *Generality* of the assurances should be improved by developing statistical methodologies for cases where the current assumptions do not hold. *Estimator performance* should be improved by accounting for the peculiarities of software testing. *Empirical studies* of the performance of various estimators should be conducted for different programs, fuzzers, and oracles. This is our vision.

REFERENCES

- [1] “Security for industrial automation and control systems - Certification of IACS supplier security policies and practices,” International Electrotechnical Commission, International Standard, 2015.
- [2] D. Halperin et al., “Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses,” in *IEEE Symposium on Security and Privacy*, ser. S&P ’08, 2008, pp. 129–142.
- [3] M. J. Harrold, “Testing: A roadmap,” in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE ’00, 2000, pp. 61–72.
- [4] A. Bertolino, “Software testing research: Achievements, challenges, dreams,” in *Future of Software Engineering*, ser. FOSE’07, 2007, pp. 85–103.
- [5] Website, “Lockheed martin webinar series: Michael whalen on the future of verification and validation.” <https://www.computer.org/cms/Computer.org/webinars/lmco/012413Slides-Whalen.pdf>, Jan. 2013, accessed: 2017-05-13.
- [6] —, “OSS-Fuzz: Five Months Later,” <https://testing.googleblog.com/2017/05/oss-fuzz-five-months-later-and.html>, 2017, accessed: 2017-11-13.
- [7] —, “Microsoft: Project springfield,” <https://www.microsoft.com/Springfield/>, 2017, accessed: 2017-11-13.
- [8] —, “Security @ adobe.” <https://blogs.adobe.com/security/tag/fuzzing>, 2017, accessed: 2017-11-13.
- [9] —, “Mozilla: Fuzzing firefox with peach.” <https://wiki.mozilla.org/Security/Fuzzing/Peach>, 2017, accessed: 2017-11-13.
- [10] M. Harman and P. O’Hearn, “From start-ups to scale-ups: Opportunities and open problems for static and dynamic program analysis,” 2018, keynote at the 18th IEEE International Working Conference on Source Code Analysis.
- [11] M. Böhme and S. Paul, “A probabilistic analysis of the efficiency of automated software testing,” *IEEE Transactions on Software Engineering*, vol. 42, no. 4, pp. 345–360, April 2016.
- [12] —, “On the efficiency of automated testing,” in *Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, ser. FSE 2014, 2014, pp. 632–642.
- [13] P. McMinn, “Search-based software test data generation: A survey: Research articles,” *Journal of Software Testing, Verification and Reliability*, vol. 14, no. 2, pp. 105–156, Jun. 2004.
- [14] J. Rushby, “Formal methods and digital systems validation for airborne systems,” NASA contractor report ; NASA CR-4551., Tech. Rep., 1993.
- [15] M. Böhme, “STADS: Software testing as species discovery,” *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 2, pp. 7:1–7:52, Jun. 2018.
- [16] Y. Basset, L. Cizek, P. Cuenoud, R. Didham, F. Guilhaumon, O. Missa, V. Novotny, F. Ødegaard, T. Roslin, J. Schmidl, A. K. Tishechkin, N. N. Winchester, D. Roubik, H.-P. Aberlenc, J. Bail, H. Barrios, J. Bridle, G. Castaño, B. Corbara, and M. Leponce, “Arthropod diversity in a tropical forest,” *Science*, vol. 338, no. 6113, pp. 1481–1484, 2012.
- [17] A. Chao and R. K. Colwell, “Thirty years of progeny from chao’s inequality: Estimating and comparing richness with incidence data and incomplete sampling,” *Statistics and Operations Research Transactions*, vol. 41, no. 1, pp. 3–54, 2017.

- [18] B. Mathis, V. Avdiienko, E. O. Soremekun, M. Böhme, and A. Zeller, "Detecting information flow by mutating input data," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE, 2017, pp. 263–273.
- [19] A. B. Wagner, P. Viswanath, and S. R. Kulkarni, "Strong consistency of the good-turing estimator," in *Proceedings of the 2006 IEEE International Symposium on Information Theory*, 2006, pp. 2526–2530.
- [20] C.-H. Zhang and Z. Zhang, "Asymptotic normality of a nonparametric estimator of sample coverage," *The Annals of Statistics*, vol. 37, no. 5A, pp. 2582–2595, 10 2009.
- [21] H. E. Robbins, "Estimating the total probability of the unobserved outcomes of an experiment," *The Annals of Mathematical Statistics*, vol. 39, no. 1, pp. 256–257, 02 1968.
- [22] A. Orlitsky and A. T. Suresh, "Competitive distribution estimation: Why is good-turing good," in *Proceedings of the 28th International Conference on Neural Information Processing Systems*, ser. NIPS'15, 2015, pp. 2143–2151.
- [23] I. J. Good, "The population frequencies of species and the estimation of population parameters," *Biometrika*, vol. 40, pp. 237–264, 1953.
- [24] T.-J. Shen, A. Chao, and C.-F. Lin, "Predicting the number of new species in further taxonomic sampling," *Ecology*, vol. 84, no. 3, pp. 798–804, 2003.
- [25] A. Chao, "Nonparametric estimation of the number of classes in a population," *Scandinavian Journal of Statistics*, vol. 11, no. 4, pp. 265–270, 1984.
- [26] —, "Estimating the population size for capture-recapture data with unequal catchability," *Biometrics*, vol. 43, no. 4, pp. 783–791, 1987.
- [27] R. K. Colwell, A. Chao, N. J. Gotelli, S.-Y. Lin, C. X. Mao, R. L. Chazdon, and J. T. Longino, "Models and estimators linking individual-based and sample-based rarefaction, extrapolation and comparison of assemblages," *Journal of Plant Ecology*, vol. 5, no. 1, p. 3, 2012.
- [28] X. Yao, M. Harman, and Y. Jia, "A study of equivalent and stubborn mutation operators using human analysis of equivalence," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, 2014, pp. 919–930.
- [29] X. Yang, Y. Chen, E. Eide, and J. Regehr, "Finding and understanding bugs in c compilers," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2011, pp. 283–294.
- [30] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, "Feedback-directed random test generation," in *Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE '07, 2007, pp. 75–84.
- [31] Website, "Afl: American fuzzy lop fuzzer," http://lcamtuf.coredump.cx/afl/technical_details.txt, 2017, accessed: 2017-11-13.
- [32] M. Böhme, V.-T. Pham, and A. Roychoudhury, "Coverage-based greybox fuzzing as markov chain," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, 2016, pp. 1032–1043.
- [33] M. Böhme, V.-T. Pham, M.-D. Nguyen, and A. Roychoudhury, "Directed greybox fuzzing," in *Proceedings of the 24th ACM Conference on Computer and Communications Security*, ser. CCS, 2017, pp. 1–16.
- [34] V.-T. Pham, M. Böhme, A. E. Santosa, A. R. Caciulescu, and A. Roychoudhury, "Smart greybox fuzzing," <https://arxiv.org/abs/1811.09447>, 2018.
- [35] M. Falk, J. Hüsler, and R.-D. Reiss, *Laws of small numbers: extremes and rare events*. Springer Science & Business Media, 2010.
- [36] E. Castillo, *Extreme value theory in engineering*. Elsevier, 2012.
- [37] P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic, "Multilevel splitting for estimating rare event probabilities," *Operations Research*, vol. 47, no. 4, pp. 585–600, 1999.
- [38] Website, "AFL: Pulling Jpegs out of Thin Air, Michael Zalewski," <https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html>, 2017, accessed: 2017-11-13.
- [39] A. Chao and S.-M. Lee, "Estimating the number of classes via sample coverage," *Journal of the American Statistical Association*, vol. 87, no. 417, pp. 210–217, 1992.
- [40] S.-M. Lee and A. Chao, "Estimating population size via sample coverage for closed capture-recapture models," *Biometrics*, vol. 50, no. 1, pp. 88–97, 1994.
- [41] A. Chao, C.-H. Chiu, R. K. Colwell, L. F. S. Magnago, R. L. Chazdon, and N. J. Gotelli, "Deciphering the enigma of undetected species, phylogenetic, and functional diversity based on good-turing theory," *Ecology*, vol. 98, no. 11, pp. 2914–2929, 2017.
- [42] M. J. Conroy, J. P. Runge, R. J. Barker, M. R. Schofield, and C. J. Fonnesebeck, "Efficient estimation of abundance for patchily distributed populations via two-phase, adaptive sampling," *Ecology*, vol. 89, no. 12, pp. 3362–3370, 2008.
- [43] W. Thompson, *Sampling rare or elusive species: concepts, designs, and techniques for estimating population parameters*. Island Press, 2013.
- [44] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.
- [45] K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov, "Addresssanitizer: A fast address sanity checker," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, 2012, pp. 28–28.
- [46] S. Elbaum and D. S. Rosenblum, "Known unknowns: Testing in the presence of uncertainty," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014, 2014, pp. 833–836.
- [47] B. D. Coleman, "On random placement and species-area relations," *Mathematical Biosciences*, vol. 54, no. 3, pp. 191–215, 1981.
- [48] S. K. Thompson, "Adaptive cluster sampling," *Journal of the American Statistical Association*, vol. 85, no. 412, pp. 1050–1059, 1990.
- [49] D. G. Horvitz and D. J. Thompson, "A generalization of sampling without replacement from a finite universe," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 663–685, 1952.