Stefan Winter

Research Statement

My research interests span all aspects of the design and assessment of complex software-intensive systems with the goal of improving their reliability. It is important to me that the approaches we develop are practical, i.e., they prove to be effective for real software and in real settings and they scale to software systems of realistic size and complexity. A number of projects that I have worked on have had a direct impact on how software is being developed and tested at large companies [29, 22, 6, 8, 9, 7]. Similarly, in the empirical studies that I have contributed to we targeted large numbers of popular and complex open source software projects to obtain insights that developers in these types of projects can benefit from [26, 3, 12, 13]. In addition to these insights, we contributed a number of bug fixes that directly improved test quality for these projects.
Striving for practical impact also implies that the output of our research must be available to researchers and practitioners. I have made the artifacts of my research publicly available whenever possible, so that others can build on my work if they desire to [30, 32, 26, 3, 24, 12, 13, 5].
In the following I first give a brief summary of my research work to date, to establish the context that shapes my ongoing and future work. I then provide an overview of the research directions I intend to follow over the coming years.

Prior Work

Fault Model Selection Effects on Software Fault Injection

Software fault injection (SFI) is a testing technique to assess the robustness of software and widely applied in the assessment of critical software, e.g., in automotive controllers or business-critical service infrastructures. To test if software systems can safely operate in adverse conditions, SFI deliberately exposes them to faulty inputs and resource-constrained execution scenarios specified by so-called *fault models*. In the work that has led to my Ph.D. dissertation I have defined a set of four comparative fault model efficiency metrics and showed in a joint study with colleagues at Microsoft Research that the results of SFI tests strongly depend on the chosen fault models [29].
The fault models classically used for SFI make the simplifying assumption that only a single fault can affect a software system at the same time. However, in a follow-up study we showed that *higher order models* composed from multiple simple faults identify robustness issues that classical models fail to detect [31].While such higher order models can be implemented with moderate effort, they result in a combinatorial explosion of *possible* fault conditions to test with, which leads to a vastly increased number of *required* tests to obtain adequate fault coverage. To mitigate the overhead that the adoption of higher order fault models entails, I proposed to increase the experiment throughput by concurrently executing experiments on parallel hardware [30]. The results show that such parallelization yields the desired throughput improvements, but also that care has to be taken in order not to threaten the reproducibility of results.
The problem of finding the right fault model, or at least identifying wrong ones, has led to a number of additional research directions. In [14, 16] we investigated the relation between bugs in programs and data corruptions on its interfaces to guide the selection of interface data corruption models. To

provide guidance for performance testers, we manually investigated a large corpus of commits in popular open source projects to study what patterns performance bugs (i.e., unnecessarily inefficient source code) take [3]. Our search for the "right bugs" has also led to a Dagstuhl seminar proposal which got accepted for August 2020 and attracted a number of internationally renowned researchers from the software engineering, fault-tolerance, and security domains. Unfortunately, it had to be canceled due to the Covid pandemic and will hopefully be revived in a different format at a later time.

Test Interference and Flaky Test Detection

The observation of possible test interferences in parallel executions has triggered further research in two main directions: (1) how such test interferences can be proactively prevented and (2) how commonly test interferences affect software testers.

Interference Prevention: To proactively prevent test interferences from affecting SFI tests, we designed a lightweight isolation approach that spawns a new process for each injected fault [25]. Due to the lightweight isolation, performance interference as in [30] were prevented and at the same time memory isolation was maintained across parallel tests. Since we noticed that the observed potential for test interference was not specific to SFI tests, we expanded our research focus to software tests in general and investigated whether we already reached an optimal trade-off between memory safety and isolation overhead with process isolation or if even weaker isolation can suffice (and provide lower isolation overhead). For this purpose we developed a lightweight static analysis to detect potential interferences between threads on files and shared heap memory for C/C++ programs [26]. The results of our study show that

- our parallel testing approach generally outperforms naive parallel test execution
- multi-threaded test execution generally does not pay off, because process handling is not much more expensive than thread handling on modern systems
- interference on shared files is a threat to naive process-parallel test execution that our approach can safely handle in many cases

Flaky Tests: Test interferences and their effects are currently being studied intensively in the software engineering community in the context of *flaky tests* [15], i.e., tests that non-deterministically pass and fail. To understand what the chances are to reproduce a certain class of flaky tests, we analyzed data from 4000 test suite executions and 4000 executions of the same tests in isolation and derived a number of practical guidelines for execution-based flaky test detection approaches [12]. In a second study, we analyzed when flaky tests become flaky, as this provides guidance on when to best run flaky test detectors [13]. We found that 75% of flaky tests are already flaky when they are introduced in the project and that another 10% of flaky tests become flaky by a commit that directly affects the test's code, which means that flaky test detection can be focused on commits that modify test code to detect a large fraction of flaky tests early.

Research Artifact Evaluations

Research artifacts commonly denote anything that has contributed to generating the results presented in a research article. To counter a potential "replicability crisis" in computer science, and specifically software engineering research, FSE pioneered research artifact evaluations in 2011. The concept has since received a healthy adoption in the software engineering and programming languages conference landscape, with ASE closing the last gap in the front row next year.

Having served on artifact evaluation committees myself and having experienced the practical challenges, I planned to scientifically investigate how well the implemented processes for artifact evaluations serve their purpose. As I quickly realized that "the purpose" of artifact evaluations is more ambiguous than I had thought, I designed a study on perceived artifact purpose together with Ben Hermann and Janet Siegmund. The results of our study showed that the perceived purpose of artifacts is two-fold, and that it depends on the community (software engineering vs. programming languages) and the artifact type (code, data, proof) [5]. Our article received the ACM SIGSOFT distinguished paper award at FSE this year.

Other Work

I had the opportunity to work on a number of research topics [20, 21, 22, 6, 8, 9, 7, 28, 2, 24], which I greatly enjoyed but have not continued due to my shift of interests. While I do not actively work in these areas now, they certainly contribute to my "systems view" on many research problems.

Future Work

The main focus of my current and future work revolves around three questions.

1. What is the impact of software bug models on software tests?
2. How can we make software tests *reliably* indicate presence/absence of bugs?
3. How can we improve reproducibility in software engineering research?

Software Bug Simulation

From my Ph.D. work I have a strong background in software fault injections and fault models. Over the last years and long discussions with colleagues in the software testing and security areas, I came to understand that the question what a bug actually is has had and still has much relevance. In software testing, fault-based adequacy or "mutation testing" is a vivid field of research with concepts that closely resemble what I have worked on for fault injections; higher order mutants [11], for instance, resemble higher order fault models in my work and what I termed the single-fault-hypothesis for SFI is conceptually close to the coupling effect hypothesis [10]. Similar concepts exist in recent software security research, in which the efficacy of ever-improving Fuzzers needs to be evaluated against a standard corpus of vulnerabilities, which has led to approaches like LAVA that automatically introduce artificial vulnerabilities into benign code [4, 23]. Three communities have been working on very similar problems, but with slightly different objectives, and I believe that a cross-community exchange will provide us with ample opportunities for high-profile research. For example, my work has largely refuted the equivalent of mutation testing's coupling effect hypothesis: Simple faults are not representative of complex faults in SFI. The most referenced work in the mutation testing area goes back to a 2005 paper [1] with an evaluation basis that would likely not withstand modern days' peer review criteria for empirical results. In fact, recent work has shown that there is only a weak correlation between common mutation adequacy measures of test suites and their (real) fault detection ability [18]. This opens the question whether mutation testing has been using the right mutation operators (as the pendant to fault models) and, more broadly, on which factors a "good" choice or design of mutation operators depends.
A second interesting direction inspired by the similarity of mutation testing and observations from my work on software fault injections is the problem of equivalent mutants (cf. [17]), i.e., variants of the program that are functionally identical to the original program. While equivalent mutants constitute a problem for mutation testing, they can be helpful for testing objectives beyond functionality. Performance testing and debugging approaches, for instance, target performance improvements of code *without changing its functionality*. Ideally, such approaches would be evaluated against functionally equivalent variants of the same code and what is a nuisance for traditional mutation testing may be highly desirable in this case.
While I do have a strong interest in fault models from other domains, such as mutation testing, my

expertise clearly lies in the fault-tolerance applications and I regularly engage in discussions with mutation testing experts (e.g., Mike Papadakis from the University of Luxembourg) for feedback on these thoughts.

Software Test Reliability

Much recent work on the reliability has focused on flaky tests. The main problem that flaky tests cause is that a usually passing test fails in a regression test execution and the developer assumes that his or her code broke the test, while there actually is a problem with the test code. From discussions with a colleague at Microsoft Research, I understood that the problem has at least one more dimension, which has remained largely unexplored. When Microsoft rolls out updates on Office365, they only gradually expand the update from one data center to the other, because they frequently encounter issues in one data center that they have not experienced in another one. The problem is similar to the problem of flaky tests: The code under test remains unchanged, but in some constellation it fails whereas it passes in others. However, there also is an important difference: Flaky tests fail or pass non-deterministically on the same setup, whereas in this case tests fail or pass deterministically for the same setup, but the test result on one setup is not representative for the test result on the other setup. There has been an early study of a similar problem in the embedded systems domain at ISSTA this year [27] and private communication with a colleague from Huawei has confirmed that it is a practical problem of significant magnitude.
I am planning to follow two different approaches to tackle the problem. First, if the differences are deterministic for each platform, there must be systematic differences that affect test outcomes and these differences do not affect the code under test, but its execution environment. Therefore, there must be effects from the execution environment on the control- and data-flow of the program under test. To capture these effects, the program's interfaces with the environment can be identified and the influence of that interface on program execution and test outcome investigated, e.g., using dynamic or static techniques, which resemble flaky test detection as in [13] or interference detection as in [26].
Second, if software tests are interpreted to resemble scientific experiments, then the aforementioned problem reads very familiar to empirical researchers: Does an obtained experimental result hold for members of a larger population than the one that has been experimented with? The problem is commonly discussed as the "external validity" of an experiment and viewed as a solved problem in some research disciplines, mostly by reference to Judea Pearl's work on transportability [19]. In my research I am planning to investigate the applicability of techniques from the domain of external validity assessment to the problem of differing software test results in different execution environments.

SE Research Reproducibility

Artifact evaluations have been introduced to virtually all major software engineering conferences by now with the goal to foster reproducible research. As scientists, we should seek evidence whether the way we conduct them indeed fulfills that goal, i.e., (1) whether research results that have undergone artifact evaluation have a higher likelihood of being reproducible than those that did not and (2) if the number of reproducible research results has increased for conferences since they have introduced artifact tracks.
Different conferences have implemented different models of artifact evaluation processes and correlations with different success rates in terms of reproducible research may indicate the knobs to turn in order to improve reproducibility. Especially if the effect of artifact evaluations on reproducibility is low, it would be fair to discuss whether the amount of time that evaluation

committees spend on reproducing research results is justified and whether that time would be better spent on assessing *reusability* of artifacts, which has been stated as a second important goal of artifact evaluations in our study [5].
As I pointed out in my discussion of research directions for reliable test results, there is a resemblance between software tests and scientific experiments and I hope that this insight can eventually also lead to better and easier artifact evaluations by applying principles and techniques from software testing (e.g., mutation testing or flaky test detection) accordingly.

References

[1] J. H. Andrews, L. C. Briand, and Y. Labiche. "Is Mutation an Appropriate Tool for Testing Experiments?" In: *Proceedings of the 27th International Conference on Software Engineering*. ICSE '05. St. Louis, MO, USA: Association for Computing Machinery, 2005, pp. 402–411. ISBN: 1581139632. DOI: 10.1145/1062455.1062530. URL: https://doi.org/10.1145/1062455.1062530.

[2] A. Chan, S. Winter, H. Saissi, K. Pattabiraman, and N. Suri. "IPA: Error Propagation Analysis of Multi-Threaded Programs Using Likely Invariants". In: *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 2017, pp. 184–195.

[3] Y. Chen, S. Winter, and N. Suri. "Inferring Performance Bug Patterns from Developer Commits". In: *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. Oct. 2019, pp. 70–81. DOI: 10.1109/ISSRE.2019.00017.

[4] B. Dolan-Gavitt, P. Hulin, E. Kirda, T. Leek, A. Mambretti, W. Robertson, F. Ulrich, and R. Whelan. "Lava: Large-scale automated vulnerability addition". In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2016, pp. 110–121.

[5] B. Hermann, S. Winter, and J. Siegmund. "Community Expectations for Research Artifacts and Evaluation Processes". In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2020. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 469–480. ISBN: 9781450370431. DOI: 10.1145/3368089.3409767. URL: https://doi.org/10.1145/3368089.3409767.

[6] T. Ishigooka, H. Saissi, T. Piper, S. Winter, and N. Suri. "Practical Use of Formal Verification for Safety Critical Cyber-Physical Systems: A Case Study". In: *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2014 IEEE International Conference on*. 2014, pp. 7–12.

[7] T. Ishigooka, F. Narisawa, K. Sakurai, N. Suri, H. Saissi, T. Piper, and S. Winter. *Method and system for testing control software of a controlled system*. US Patent 9,575,877. Feb. 2017.

[8] T. Ishigooka, H. Saissi, T. Piper, S. Winter, and N. Suri. "Practical Formal Verification for Model Based Development of Cyber-Physical Systems". In: *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*. Aug. 2016, pp. 1–8. DOI: 10.1109/CSE-EUC-DCABES.2016.154.

[9] T. Ishigooka, H. Saissi, T. Piper, S. Winter, and N. Suri. "Safety Verification Utilizing Model-based Development for Safety Critical Cyber-Physical Systems". In: *JIP* 25 (2017), pp. 797–810.

[10] Y. Jia and M. Harman. "An analysis and survey of the development of mutation testing". In: *IEEE transactions on software engineering* 37.5 (2010), pp. 649–678.

[11] Y. Jia and M. Harman. "Higher Order Mutation Testing". In: *Information and Software Technology* 51.10 (2009). Source Code Analysis and Manipulation, SCAM 2008, pp. 1379–1393. ISSN: 0950-5849. DOI: `https://doi.org/10.1016/j.infsof.2009.04.016`. URL: `http://www.sciencedirect.com/science/article/pii/S0950584909000688`.

[12] W. Lam, S. Winter, A. Astorga, V. Stodden, and D. Marinov. "Understanding Reproducibility and Characteristics of Flaky Tests Through Test Reruns in Java Projects". In: *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 2020, pp. 403–413. DOI: `10.1109/ISSRE5003.2020.00045`.

[13] W. Lam, S. Winter, A. Wei, T. Xie, D. Marinov, and J. Bell. "A Large-Scale Longitudinal Study of Flaky Tests". In: *Proc. ACM Program. Lang.* 4.OOPSLA (Nov. 2020). DOI: `10.1145/3428270`. URL: `https://doi.org/10.1145/3428270`.

[14] A. Lanzaro, R. Natella, S. Winter, D. Cotroneo, and N. Suri. "An Empirical Study of Injected Versus Actual Interface Errors". In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ISSTA 2014. 2014, pp. 397–408.

[15] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov. "An empirical analysis of flaky tests". In:

[16] R. Natella, S. Winter, D. Cotroneo, and N. Suri. "Analyzing the Effects of Bugs on Software Interfaces". In: *IEEE Transactions on Software Engineering* 46.3 (Mar. 2020), pp. 280–301. ISSN: 2326-3881. DOI: `10.1109/TSE.2018.2850755`.

[17] M. Papadakis, Y. Jia, M. Harman, and Y. Le Traon. "Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. IEEE. 2015, pp. 936–946.

[18] M. Papadakis, D. Shin, S. Yoo, and D.-H. Bae. "Are Mutation Scores Correlated with Real Fault Detection? A Large Scale Empirical Study on the Relationship between Mutants and Real Faults". In: *Proceedings of the 40th International Conference on Software Engineering*. ICSE '18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 537–548. ISBN: 9781450356381. DOI: `10.1145/3180155.3180183`. URL: `https://doi.org/10.1145/3180155.3180183`.

[19] J. Pearl and E. Bareinboim. "External validity: From do-calculus to transportability across populations". In: *Statistical Science* (2014), pp. 579–595.

[20] T. Piper, S. Winter, P. Manns, and N. Suri. "Instrumenting AUTOSAR for dependability assessment: A guidance framework". In: *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*. 2012, pp. 1–12.

[21] T. Piper, S. Winter, O. Schwahn, S. Bidarahalli, and N. Suri. "Mitigating Timing Error Propagation in Mixed-Criticality Automotive Systems". In: *Real-Time Distributed Computing (ISORC), 2015 IEEE 18th International Symposium on*. 2015, pp. 102–109.

[22] T. Piper, S. Winter, N. Suri, and T. E. Fuhrman. “On the Effective Use of Fault Injection for the Assessment of AUTOSAR Safety Mechanisms”. In: *Proceedings of the European Dependable Computing Conference (EDCC)*. 2015.

[23] S. Roy, A. Pandey, B. Dolan-Gavitt, and Y. Hu. “Bug synthesis: Challenging bug-finding tools with deep faults”. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2018, pp. 224–234.

[24] H. Saissi, S. Winter, O. Schwahn, K. Pattabiraman, and N. Suri. “TraceSanitizer - Eliminating the Effects of Non-Determinism on Error Propagation Analysis”. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2020, pp. 52–63.

[25] O. Schwahn, N. Coppik, S. Winter, and N. Suri. “FastFI: Accelerating Software Fault Injections”. In: *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*. Dec. 2018, pp. 193–202. DOI: 10.1109/PRDC.2018.00035.

[26] O. Schwahn, N. Coppik, S. Winter, and N. Suri. “Assessing the State and Improving the Art of Parallel Testing for C”. In: *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2019. Beijing, China: ACM, 2019, pp. 123–133. DOI: 10.1145/3293882.3330573.

[27] P. E. Strandberg, T. J. Ostrand, E. J. Weyuker, W. Afzal, and D. Sundmark. “Intermittently Failing Tests in the Embedded Systems Domain”. In: *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2020. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 337–348. ISBN: 9781450380089. DOI: 10.1145/3395363.3397359. URL: https://doi.org/10.1145/3395363.3397359.

[28] F. Tan, L. Liu, S. Winter, Q. Wang, N. Suri, L. Bu, Y. Peng, X. Liu, and X. Peng. “Cross-Domain Noise Impact Evaluation for Black Box Two-Level Control CPS”. In: *ACM Trans. Cyber-Phys. Syst.* 3.1 (Sept. 2018), 2:1–2:25.

[29] S. Winter, C. Sarbu, N. Suri, and B. Murphy. “The impact of fault models on software robustness evaluations”. In: *Software Engineering (ICSE), 2011 33rd International Conference on*. 2011, pp. 51–60.

[30] S. Winter, O. Schwahn, R. Natella, N. Suri, and D. Cotroneo. “No PAIN, No Gain? The Utility of PArallel Fault INjections”. In: *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*. 2015, pp. 494–505.

[31] S. Winter, M. Tretter, B. Sattler, and N. Suri. “simFI: From single to simultaneous software fault injections”. In: *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*. 2013, pp. 1–12.

[32] S. Winter, T. Piper, O. Schwahn, R. Natella, N. Suri, and D. Cotroneo. “GRINDER: On Reusability of Fault Injection Tools”. In: *Proceedings of the 10th International Workshop on Automation of Software Test*. AST ’15. 2015, pp. 75–79.