Research Statement

Cindy Rubio González
University of California, Davis

Given the ubiquity of software in modern life, software defects (also known as bugs) have a major impact on the economy, on safety, and on the quality of life. The diagnosis and repair of software bugs consumes a large proportion of the time and budgets of software teams in public and private institutions.

I am an Assistant Professor in the Department of Computer Science at UC Davis since November 2014. My research spans the areas of Programming Languages (PL) and Software Engineering (SE), with a focus on program analysis for automated bug finding and program optimization. Program analysis is the process of automatically analyzing programs to discover facts about their behavior. In particular, I develop analysis techniques to improve the reliability and performance of systems software and numerical applications. Most recently, I have also become interested in the automatic creation of large-scale datasets for SE research, and the application of Machine Learning techniques to solve SE problems. In all these areas, *the goal of my research group is to devise new algorithms and develop tools that we apply to real-world code*. Below I summarize these main areas of research.

1 Analyzing Error Handling in Large Software Systems

Error handling is the process of detecting and responding to the occurrence of runtime errors in software. Bugs in software error handlers are some of the most pervasive, dangerous, and difficult to detect bugs. These bugs can lead to problems such as system crashes, security vulnerabilities, silent data loss, and data corruption. My research aims at developing *precise and scalable program analyses* to understand, find, and fix error-handling bugs in *large software systems*. In particular, we focus on finding bugs in the propagation of error codes in C applications, bugs in the way software checks for errors, and bugs in the manner software handles errors.

My contributions include static analysis tools [6, 7, 8, 12, 15, 16, 17, 24] to find error-handling bugs. My work is the first to find bugs related to the incorrect propagation of error codes, and had made a real-world impact: my tools have found 452 previously unknown bugs in Linux, a critical bug in heavily tested code used for space missions by the NASA/JPL Laboratory, 447 previously unknown bugs in MPI libraries, 164 previously unknown bugs in OpenSSL and other high-profile C applications, and several high-impact security vulnerabilities in the Firefox web browser. I have given over 25 invited talks on this line of work at places such as UC Berkeley, Stanford University, EPFL, Microsoft Research, Google, Mozilla, and Los Alamos National Laboratory. This work has been funded by an NSF CRII (Research Initiation Initiative) grant for junior faculty. My Ph.D. student Daniel DeFreez won the first place in the Student Research Competition at the Foundations of Software Engineering (FSE'18) in the graduate category for his work on mining error-handling specifications [5, 8].

2 Improving the Reliability and Performance of Numerical Software

From machine learning to safety-critical systems, a large variety of applications today make use of numerical software, i.e., software used to perform numerical calculations. Floating point is a widely used representation that approximates real numbers. By nature, floating point introduces imprecision in numerical calculations, which has led to software bugs that have caused catastrophic failures. The goal of my research is to develop analysis techniques and tools to (1) optimize the use of floating point in programs to decrease their running time, energy consumption, and/or memory usage, and (2) improve the reliability of numerical code by finding and fixing bugs due to the imprecision of floating point.

In the area of precision tuning, I have developed tools [13, 18, 19] that improve the performance of numerical programs while still meeting accuracy constraints. We have shown that by effectively using mixed floating-point precision, we are able to speedup programs up to 40%. Our tool Precimonious [19] is the first to automatically tune floating-point programs. Precimonious conducts a search over the types of floating-point variables to find a type configuration that improves performance given an error threshold. Later we developed Blame Analysis [18], a shadow execution based technique that speeds up tuning while solely focusing on precision. Unlike previous work that employs black-box tuning techniques, Hierarchical FPTuner (our most recent tool) [13] leverages the dependencies among floating-point variables in the program to make a choice of precision that leads to higher performance speedups 60% faster than previous approaches.

In the area of bug finding, we conducted the first empirical study of real-world numerical bugs [10] in widely-used numerical libraries, including NumPy, SciPy, LAPACK, and the GNU Scientific Library. We studied 269 known

numerical bugs, and investigated their nature and how programmers fix these bugs. Most recently, we developed a a symbolic-execution based tool named FPGen [14] that generates floating-point inputs that maximize numerical error, and a framework for differential testing of numerical libraries [23].

I have given over 20 invited talks on this topic at places such as MIT, Oracle, and Lawrence Berkeley National Laboratory, and co-organized a Dagstuhl seminar on the Analysis and Synthesis of Floating-Point Programs. I am a co-founder and co-organizer of the International Workshop on Software Correctness for HPC Applications (in conjunction with SuperComputing), which we are running for a third consecutive year. I am co-organizing an ICERM workshop on Variable Precision in Mathematical and Scientific Computing to be held in May 2020. This work is supported by a DOE Early Career Award, an NSF CAREER Award, and a Hellman Fellowship.

3 Creating Large-Scale Datasets for Software Engineering Research

PL/SE research on bug finding and automated program repair is empirically grounded in datasets of bugs. Because the ultimate goal is to analyze real-world software, it is critical for the bugs in these datasets to represent *real bugs*. Furthermore, it is imperative for the bugs to be reproducible, i.e., if one runs the buggy program, we should observe a software failure. Recently, we designed BugSwarm [21], an infrastructure to create a large-scale dataset of *reproducible* failures and fixes mined from open-source projects. The BugSwarm dataset has an unprecedented size and diversity (types of failures, build systems, etc.), while retaining sufficient fidelity of detail to allow the replication of the failures. Because its creation is fully automated, BugSwarm is designed to continuously grow, allowing not only to increase its size over time, but to capture new technologies as software evolves. BugSwarm currently consists of over 3,000 reproducible failures an fixes from open-source projects written in Java and Python, "packaged" in Docker images. We believe that BugSwarm will impact how analysis tools are designed and evaluated as well as open new opportunities in the creation of domain-specific benchmarks at scale. BugSwarm is currently funded by an NSF CRI (Research Infrastructure) grant.

My student David Tomassi won the first place in the Student Research Competition at the Foundations of Software Engineering (FSE'18) in the undergraduate category for his study on the effectiveness of bug detectors using BugSwarm artifacts [20]. Recently, we conducted a study of the effectiveness of bug detectors (SpotBugs [2], Infer [1], and NullAway [3]) at identifying over 100 real-world null pointer dereferences found in the BugSwarm dataset [22]. I recently received a Facebook Testing and Verification Award to continue this work.

I am also interested in the usability of program analysis tools. I published a short paper that explores this topic, and proposes a framework to compile tools to JavaScript for easy and fast tool evaluation on the web [11]. We have also developed a tool named GitcProc for processing and classifying GitHub commits based on characteristics of interest such as specific language constructs changed when fixing a bug [4].

4 Applying Machine Learning for Software Engineering

An emerging area of research is machine learning for software engineering. I am currently working on combining program analysis and machine learning for program understanding and bug finding. In our recent work [8, 9], we used neural networks to learn a distributed representation of functions in the Linux kernel. We developed Func2vec, a technique for learning a function embedding that can be used to identify function synonyms. A *function embedding* is an embedding $\Phi : \mathtt{F} \rightarrow \mathbb{R}^d$ that maps each function $\mathtt{f} \in \mathtt{F}$ in the program to a d-dimensional vector in $\mathbb{R}^d$ such that vectors for function synonyms are in close proximity. *Function synonyms* are functions that serve the same purpose or play a similar role in code. Furthermore, we developed a tool named EHnfer that mines error handling specifications. We showed that leveraging function synonyms in the mining process enables mining error-handling specifications across multiple implementations of Linux file systems and drivers.

Current and future work includes creating program representations to find a variety of bugs such as Null Pointer Dereferences, which cause program crashes. I gave a Keynote at the Machine Learning for Software Engineering (ML4SE) Workshop in Montreal Canada describing this work, and we recently received a Facebook Probability and Programming Research Award to continue this line of work.

References

[1] Infer. http://fbinfer.com/, 2019.

[2] SpotBugs. https://spotbugs.github.io/, 2019.

[3] S. Banerjee, L. Clapp, and M. Sridharan. Nullaway: Practical type-based null safety for java. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pages 740–750, New York, NY, USA, 2019. ACM.

[4] C. Casalnuovo, Y. Suchak, B. Ray, and C. Rubio-González. Gitcproc: a tool for processing and classifying github commits. In T. Bultan and K. Sen, editors, *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, Santa Barbara, CA, USA, July 10 - 14, 2017*, pages 396–399. ACM, 2017.

[5] D. DeFreez. Mining error-handling specifications for systems software. In *ESEC/SIGSOFT FSE*, pages 983–985. ACM, 2018.

[6] D. DeFreez, H. M. Baldwin, C. Rubio-González, and A. V. Thakur. Effective error-specification inference via domain-knowledge expansion. In *ESEC/SIGSOFT FSE*, pages 466–476. ACM, 2019.

[7] D. DeFreez, A. Bhowmick, I. Laguna, and C. Rubio-González. Detecting and Reproducing Error-Code Propagation Bugs in MPI Implementations. Under Submission, 2019.

[8] D. DeFreez, A. V. Thakur, and C. Rubio-González. Path-based function embedding and its application to error-handling specification mining. In *ESEC/SIGSOFT FSE*, pages 423–433. ACM, 2018.

[9] D. DeFreez, A. V. Thakur, and C. Rubio-González. Path-based function embeddings. In *ICSE (Companion Volume)*, pages 430–431. ACM, 2018.

[10] A. D. Franco, H. Guo, and C. Rubio-González. A comprehensive study of real-world numerical bug characteristics. In *ASE*, pages 509–519. IEEE Computer Society, 2017.

[11] J. Galenson, C. Rubio-González, S. Chasins, and L. Gong. Research.js: Evaluating research tool usability on the web. In J. Sunshine, T. D. LaToza, and C. Anslow, editors, *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools, Portland, OR, USA, October 21, 2014*, pages 53–54. ACM, 2014.

[12] H. S. Gunawi, C. Rubio-González, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and B. Liblit. EIO: Error Handling is Occasionally Correct. In M. Baker and E. Riedel, editors, *FAST*, pages 207–222. USENIX, 2008.

[13] H. Guo and C. Rubio-González. Exploiting community structure for floating-point precision tuning. In *ISSTA*, pages 333–343. ACM, 2018.

[14] H. Guo and C. Rubio-González. Effective Generation of Error-Inducing Floating-Point Inputs via Symbolic Execution. Under Submission, 2019.

[15] C. Rubio-González, H. S. Gunawi, B. Liblit, R. H. Arpaci-Dusseau, and A. C. Arpaci-Dusseau. Error Propagation Analysis for File Systems. In M. Hind and A. Diwan, editors, *PLDI*, pages 270–280. ACM, 2009.

[16] C. Rubio-González and B. Liblit. Expect the Unexpected: Error Code Mismatches Between Documentation and the Real World. In S. Lerner and A. Rountev, editors, *PASTE*, pages 73–80. ACM, 2010.

[17] C. Rubio-González and B. Liblit. Defective Error/Pointer Interactions in the Linux Kernel. In M. B. Dwyer and F. Tip, editors, *ISSTA*, pages 111–121. ACM, 2011.

[18] C. Rubio-González, C. Nguyen, B. Mehne, K. Sen, J. Demmel, W. Kahan, C. Iancu, W. Lavrijsen, D. H. Bailey, and D. Hough. Floating-point precision tuning using blame analysis. In *ICSE*, pages 1074–1085. ACM, 2016.

[19] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough. Precimonious: Tuning Assistant for Floating-Point Precision. In W. Gropp and S. Matsuoka, editors, *SC*, page 27. ACM, 2013.

[20] D. A. Tomassi. Bugs in the wild: examining the effectiveness of static analyzers at finding real-world bugs. In *ESEC/SIGSOFT FSE*, pages 980–982. ACM, 2018.

[21] D. A. Tomassi, N. Dmeiri, Y. Wang, A. Bhowmick, Y. Liu, P. T. Devanbu, B. Vasilescu, and C. Rubio-González. Bugswarm: mining and continuously growing a dataset of reproducible failures and fixes. In *ICSE*, pages 339–349. IEEE / ACM, 2019.

[22] D. A. Tomassi and C. Rubio-González. On the Real-World Effectiveness of Static Bug Detectors at Finding Null Pointer Exceptions. Under Submission, 2019.

[23] J. Vanover, X. Deng, and C. Rubio-González. Discovering Discrepancies in Numerical Libraries. Under Submission, 2019.

[24] C. Weiss, C. Rubio-González, and B. Liblit. Database-backed program analysis for scalable error propagation. In *ICSE (1)*, pages 586–597. IEEE Computer Society, 2015.