Research Statement

Gregory James Gay
Department of Computer Science & Engineering, University of South Carolina
3A66 Swearingen Engineering Center, 315 Main Street, Columbia, SC 29208
(803)-777-9479, greg@greggay.com

1. Research Overview

My research interests lie in the field of **software engineering**, with an emphasis on **software testing**. In my work, I harness the information content of software development artifacts in order to improve the quality and ease the human burden of the testing process. Many of my approaches and innovations are rooted in the emerging field of data analytics—a **data-centric approach** that forms an intersection between search, optimization, data mining, and large-scale empirical investigation.

A few of my contributions to date include:

- Enabling the use of models to judge test results for real-time embedded systems, including methods to prioritize and filter fault-indicative behaviors.
- Insight into the limitations of using structural coverage metrics to guide test case generation, as well as a new structural coverage metric designed to overcome such limitations.
- Search techniques that select optimal system monitoring points and optimize early-lifecycle decision making.
- Learning methods that identify critical design flaws and filter cross-company project information for local use.
- A recommendation engine that locates software faults through feedback-assisted source code search.

My work to date has resulted in:

- Sixteen refereed publications—with four more currently under review—in some of the most important journals, conferences, and workshops in the software engineering field.
- A three-year National Science Foundation graduate research fellowship.
- Fruitful research collaborations with NASA's Jet Propulsion Lab and Ames Research Center, and the Lab for Internet Software Technologies at the Chinese Academy of Sciences.

I advocate the cause of open science, and make great efforts to disseminate my work to the broader community. My tools, data, and even experimental infrastructure are—when possible—released under open source licenses. I strongly believe that the results of research work should be released publicly and transparently so that society may benefit from my work and so that other researchers may replicate, improve, and even refute my results.

2. Past Research

I have a large body of experience in applying data-centric approaches to software engineering concerns—deriving information from software artifacts, performing extensive experimentation to derive the theories underlying these observations, and—ultimately—advancing towards solutions for the research problems studied:

- I designed new search algorithms that balance competing factors in early-lifecycle requirements models to produce optimal solutions to the difficult design decisions made by engineers at NASA's Jet Propulsion Laboratory [3]. My methods outperformed state-of-the-art optimization techniques in terms of both speed and result quality [8]. Further work argued on both an experimental and theoretical basis for the use of this algorithm as a baseline method for other search-based software engineering problems [2].
- I crafted a recommendation system that—learning from project artifacts—locates the blocks of source code where a fault may reside [4]. The tester could then offer feedback on the solutions, and based on that feedback, the system returns new candidate code blocks.
- While working at the NASA Ames Research Center, I designed a new algorithm for *treatment learning*—the process of identifying the specific factors leading to a result classification—and applied this algorithm to NASA simulation data to identify design defects in their software projects [7].
- As a visiting researcher at the Chinese Academy of Sciences' Lab for Internet Software Technologies, I worked on two projects related to the challenges of concurrently drawing useful information from the project data of multiple companies. I co-designed an analogy-based estimation filter that improved conclusion validity for

cross-company datasets [15]. I also assisted with designing a metric for measuring the heterogeneity of the cross-company data—a useful metric for selecting optimal data handling policies [1].

- While part of a broader project on the information content of software project metrics, I discovered evidence for a *performance ceiling* in the quality of conclusions derived from defect prediction data [16]. Experimental results indicated that as few as fifty training instances yielded as much information as larger training sets, hinting that rather than improving algorithms, improvements were needed in the data collected.
- The need for improvements in data collection, experiment repeatability, and result dissemination led me to invent a toolkit for the creation and sharing of experiments [6]. Using this kit, I was able to replicate another researchers work and then share the entire experiment—including source code, algorithms, and data—with the broader research community. I later worked with another student to expand and improve this toolset [17].

3. Current Research Agenda

Software testing is a crucial development activity—we must be able to rely on the systems we build to produce correct results. The ever-increasing complexity of software is making it both more difficult and expensive to ensure the correctness of system behavior. My research work improves the practice of software testing—optimizing result quality, efficiency, and cost—through improvements in the selection, design, and automation of the artifacts of the testing process. In particular, I am interested in the *test oracle*, an artifact that judges whether or not the behavior of the system is acceptable. One of the most difficult problems being studied in software testing research is the challenge of building test oracles that can automatically judge the results of a wide variety of testing scenarios.

3.1. The Influence of Testing Artifacts on Fault Identification

It is impossible to prove the absence of faults in software; therefore, it is important to both *build a test suite that will drive the system to encounter a fault* and ensure that such faults will *be observed and identified by the test oracle*. To that end, I investigate the influence that different testing artifacts—such as the oracle, test inputs, and test quality metrics—and different test generation techniques have on the quality of the overall testing process.

Test Suite Optimization. Measurements of the amount of source code exercised by tests are used to estimate the adequacy of a test suite, and are required for safety certification of avionics systems. Given the central role of such measurements, they are often used as a goal target for automated test generation methods. I examined the use of structural coverage criteria as both a test generation target and as a supplement to existing forms of testing [20]. Empirical investigation indicated that code coverage alone is a poor indicator of the likelihood of identifying faults. The use of structural coverage metrics as the optimization goal for test generation resulted in a "shortest-path approach" to test creation, leading to tests that were either unable to drive the system towards incorrect behavior or that failed to make those behaviors observable by the test oracle. Expanded work found that these results hold for a wide variety of systems and coverage metrics and offers recommendations into how to improve automated test generation to account for such weaknesses [14], [13], [18].

Following this work, we proposed a new coverage metric that requires both (a) exercising the source code, and (b), guaranteeing that the result of that execution is observed by the test oracle [21]. When used as a test generation target, test suites built to meet this new metric had better fault detection capabilities and were less sensitive to program structure than other metrics.

Test Oracle Optimization. Past research yielded evidence that—just as test selection influences the quality of the testing process—the selection of the program space and behaviors that the test oracle monitors and judges impacts the subsequent detection of faults. Motivated by this observation, I designed an algorithm that automatically learns which variables should be monitored and evaluated by the test oracle [19]. These optimized oracles more effectively identify faults than oracles built using commonly employed industrial practices. I later expanded this work by identifying the scenarios where this automated selection practice is appropriate, which types of test suites were more effective in training these oracles, and how much training data is needed to yield effective results [12].

3.2. Model-Based Test Oracles

A promising solution to the test oracle problem is to build a behavioral model of the system—a practice commonly employed in safety-critical systems for analysis purposes—and use the model as an oracle when testing. Such models

prescribe behaviors that should be exhibited in response to a particular input, and if the behaviors exhibited by the system conform to those prescribed by the model, then the oracle judges the system to be acting correctly.

My current research focus is on the use of behavioral models as test oracles for real-time and embedded systems. As these models represent an abstracted view of the system, they may not reflect the actual conditions that the system operates under. Abstraction makes the use of these models as test oracles challenging, as the behavior of a real-time system depends on both the values of inputs and outputs and their time of occurrence. When executed on a real hardware platform, the system under test may exhibit behavior that differs from what the model predicts for a given input. Over time, these differences can build to the point where the oracle and the system have diverged into different execution paths, leading to a "failure" verdict even when the system is still exhibiting acceptable behavior.

My work has enabled the use of such models as test oracles by steering the model—when behaviors begin to diverge, a search process automatically adjusts the execution of the model to try and match the behavior of the system, within a set of constraints. Addressing this problem has required research work in a number of areas:

- Comparison metrics to quantify behavioral differences and guide the steering process.
- Search methods that best steer the model towards matching the behavior of the system.
- Policies to limit steering to behaviors that are only possible without violating the system requirements, including guidance on how to derive constraints from domain expertise and learning algorithms to automatically extract and strengthen constraints using test cases with known outcomes.

Steering significantly increases test oracle accuracy and, thus, has significant implications for reducing development costs and improving fault detection [11], [10], [9]. The use of this steering framework can allow developers to focus on behavioral difference indicative of real faults, rather than spending time examining test failure verdicts that can be blamed on an excessively rigid oracle model.

4. Future Research Plans

My long-range goal is to improve the practice of software testing through a number of initiatives, including further study of the factors that influence test quality, new techniques for the generation of effective tests, and new innovations in the development, design, and automation of the test oracle. Such research efforts will lead to improvements in testing practices and a lessening of the burden on the human tester. Some of the research challenges that I plan to address include:

- Automated synthesis of software test oracles—it may be possible to automatically create oracles by mining the information content of software development artifacts.
- Automated oracle debugging—as with any artifact, it is possible to build an incorrect test oracle. I will examine ways to automatically debug the oracle by learning from test executions and other sources of information.
- Optimization of execution monitoring—given the expense of system monitoring, I seek methods to sort and prioritize testing goals in order to identify the minimal set of execution points that need to be monitored in order to observe and identify faults.
- Test suite optimization—the oracle and test suite mutually influence the effectiveness of the other. I plan to further examine methods of test selection, test suite prioritization, and test generation to enhance the fault-identification abilities of test oracles.

The field of software testing is filled with difficult problems and exciting challenges—it is a field where I can draw conclusions and make recommendations from large repositories of project information, derive general theories and techniques from such data, and use such techniques and theories to advance the state of the art.

While my focus is currently on the challenges of software testing, I also continue to argue for better data collection procedures and propose methods to better use the data that we can collect. To this end, I recently proposed the use of recommendation systems as the basis for community-assisted project recommendations [5]. I will also continue to undertake research efforts to address problems related to data quality and work towards the creation of improved data analytic techniques.

References

[1] J. Chen, Y. Yang, W. Zhang, and G. Gay. Measuring the heterogeneity of cross-company dataset. In *Proceedings of the 11th International Conference on Product Focused Software*, PROFES '10, pages 55–58, New York, NY, USA, 2010. ACM.

[2] G. Gay. A baseline method for search-based software engineering. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, PROMISE '10, pages 2:1–2:11, New York, NY, USA, 2010. ACM.

[3] G. Gay. The robust optimization of non-linear requirements models, 2010.

[4] G. Gay, S. Haiduc, A. Marcus, and T. Menzies. On the use of relevance feedback in IR-based concept location. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 351 –360, sept. 2009.

[5] G. Gay and M. Heimdahl. Community-assisted software engineering decision making. In *International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE'13), Over the Horizon Track*, 2013. Available as UMN tech report 13-015, http://www.cs.umn.edu/research/technical_reports/view/13-015.

[6] G. Gay, T. Menzies, B. Cukic, and B. Turhan. How to build repeatable experiments. In *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, PROMISE '09, pages 15:1–15:9, New York, NY, USA, 2009. ACM.

[7] G. Gay, T. Menzies, M. Davies, and K. Gundy-Burlet. Automatically finding the control variables for complex system behavior. *Automated Software Engg.*, 17(4):439–468, Dec. 2010.

[8] G. Gay, T. Menzies, O. Jalali, G. Mundy, B. Gilkerson, M. Feather, and J. Kiper. Finding robust solutions in requirements models. *Automated Software Engg.*, 17(1):87–116, Mar. 2010.

[9] G. Gay, R. Rayadurgam, and M. Heimdahl. Automated steering of model-based test oracles to admit real program behaviors. *To be submitted in December 2014*, 2014. Draft available on request.

[10] G. Gay, S. Rayadurgam, and M. P. Heimdahl. Improving the accuracy of oracle verdicts through automated model steering. In *Proceedings of the 29th IEEE/ACM International Conference on Automated Software Engineering (ASE 2014)*, ASE '14, New York, NY, USA, 2014. ACM.

[11] G. Gay, S. Rayadurgam, and M. P. Heimdahl. Steering model-based oracles to admit real program behaviors. In *Proceedings of the 36th International Conference on Software Engineering – NIER Track*, ICSE '14, New York, NY, USA, 2014. ACM.

[12] G. Gay, M. Staats, M. Whalen, and M. Heimdahl. Automated oracle data selection support. *Under revision for IEEE Transactions on Software Engineering*, 2014. Draft available on request.

[13] G. Gay, M. Staats, M. Whalen, and M. Heimdahl. The risks of coverage-directed test case generation. *Under revision for IEEE Transactions on Software Engineering*, 2014. Draft available on request.

[14] G. Gay, M. Staats, M. W. Whalen, and M. P. E. Heimdahl. Moving the goalposts: Coverage satisfaction is not enough. In *Proceedings of the 7th International Workshop on Search-Based Software Testing*, SBST 2014, pages 19–22, New York, NY, USA, 2014. ACM.

[15] E. Kocaguneli, G. Gay, T. Menzies, Y. Yang, and J. W. Keung. When to use data from other projects for effort estimation. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 321–324, New York, NY, USA, 2010. ACM.

[16] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang. Implications of ceiling effects in defect predictors. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, PROMISE '08, pages 47–54, New York, NY, USA, 2008. ACM.

[17] A. Nelson, T. Menzies, and G. Gay. Sharing experiments using open-source software. *Softw. Pract. Exper.*, 41(3):283–305, Mar. 2011.

[18] A. Rajan, G. Gay, M. Staats, M. Whalen, and M. Heimdahl. Automated steering of model-based test oracles to admit real program behaviors. *To be submitted in December 2014*, 2014. Draft available on request.

[19] M. Staats, G. Gay, and M. Heimdahl. Automated oracle creation support, or: how I learned to stop worrying about fault propagation and love mutation testing. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 870–880, 2012.

[20] M. Staats, G. Gay, M. Whalen, and M. Heimdahl. On the danger of coverage directed test case generation. In *Proceedings of the 15th international conference on Fundamental Approaches to Software Engineering*, FASE'12, pages 409–424, Berlin, Heidelberg, 2012. Springer-Verlag.

[21] M. Whalen, G. Gay, D. You, M. Heimdahl, and M. Staats. Observable modified condition/decision coverage. In *Proceedings of the 2013 International Conference on Software Engineering*. ACM, May 2013.