Research Statement: I am interested in increasing software quality by reducing the cost of maintenance through automated defect localization and repair.

**A Human Study of Fault Localization Accuracy (Intl Conf Software Maintenance 2010)**
As a first step in this line of research, I proposed to understand what makes debugging a particularly difficult and time-consuming task for humans. We conducted a human study that quantitatively and qualitatively measured the difficulty associated with maintenance tasks in which 65 participants were asked to locate bugs (or show the absence thereof) within a given source file. We investigated four categories of features that contribute to debugging difficulty: the type of defect present, syntactical features of the associated code, contextual features of the code, and various design choices related to abstractions in the code. Whereas existing software quality metrics generally focus on only one of these areas, we found that features from all four categories influenced debugging difficulty. We derived a static source code quality metric that correlates with the measured difficulty between three and five times more than existing software quality metrics. In practice, such a metric could be used by static analysis tools or in code reviews to measure “future maintainability” or “debugging difficulty” of a given piece of code.

**Fault Localization Using Textual Similarities (under submission – ECOOP 2011)**
Our human study started with bugs that had been localized to a single file, but the size and complexity of modern systems complicates such fault localization considerably. To further reduce the costs of software maintenance, we desired a lightweight, accurate fault localization technique. Existing approaches require dynamic runtime information or developer intervention. We instead leverage the natural language information inherent in both the code base and available defect reports, thus requiring no runtime information. We developed a structural similarity metric between defect reports and source code that was able to eliminate 91% of the code search space, on average, when attempting to locate an individual defect, as measured on over five thousand real defects from OpenOffice, Mozilla and Eclipse. We further demonstrate that our technique is an improvement over both basic code search baselines and also existing fault localization techniques. While existing techniques can pinpoint code that is directly implicated in a failing test case, our technique is useful in detecting code that must be fixed but is not directly implicated by a test case.

**Automated Hardening via Evolutionary Processes and Proactive Diversity**
Much existing work focuses on fixing bugs after they are detected. We instead propose a proactive measure, arguing that automatically hardening systems against potential sources of error prior to detection will facilitate the overall maintenance process and improve software quality in general. We hypothesize that by creating a diverse and yet functional population of program variants, to be run in tandem, we will be able to detect previously unidentified faults and create fixes automatically based on output or system call differences between program variants. This approach builds on our existing work involving automated program repair but introduces new challenges in terms of promoting runtime diversity and selecting variants that include functionality that might fix unknown bugs. While this work is an ongoing, long-term research project, we have already explored creating proactively functional variants by using the output of Coverity’s static analysis tool as well as promoting diversity using various syntactical and contextual difference algorithms. In preliminary results, given an off-the-shelf program with ten unknown, adversary-seeded bugs, we are able to produce ten program variants, one of which, on average, detects at least one unknown bug, as well as providing a starting point patch for repairing it.