Research Statement

Muhammad Zubair Malik

I am interested in *Software Engineering*, more specifically in *Software Verification, Validation and Testing*. The goal of my research is to improve *software reliability and security* in a cost effective manner. I want to leverage the advances in verification, constraint-solving, cloud-computing, big-data and machine-learning for improving techniques in program transformation, program understanding, security and automated debugging.

Overview

Software bugs are expensive, costing companies billions of dollars in repairs, lawsuits and lost sales. In 2003, NIST estimated that the annual cost of software bugs to the US economy was 60 billion dollars. A more recent study by University of Cambridge (UK) in 2013 puts this estimate at 312 billion dollars annually, with an estimated 50% of developers' time wasted in debugging. Coding errors not only cause financial loss and physical damage but in the worst cases can result in fatalities. There is an urgent need to develop better techniques to detect, understand, mitigate and remove bugs quickly. My research focuses on tackling these problems. I have devised approaches: 1) to suggest likely bug fixes to automate debugging [1, 2], 2) generate program invariants to improve code understanding [3, 4, 5], and 3) devised techniques to improve efficiency of run-time error recovery [6]. I believe that the next generation of Integrated Development Environments will be equipped with such powerful automated debugging, analysis and synthesis tools to assist developers. This will make programming more accessible, reduce development time and make software more reliable.

Current and Previous Work

Automated Debugging

Using a traditional debugging environment, a programmer has to manually trace the execution of the program. On finding a corrupted program state, the programmer has to make assumptions about fault locations and create possible fixes. Not only is this time consuming, rather the fixes may introduce some new bugs. If we can synthesize a correct version of the same program with minimal difference, then we can use the corrective change to understand, explain and localize the bug. I devised such an approach for programs manipulating complex data structures that pervade modern software [1, 2].

Systems with high reliability and availability requirements have used data structure repair over the last few decades as an effective means to recover on-the-fly from errors in program state. We developed the insight that: since the goal of repair is to transform an erroneous state into an acceptable state, the state mutations performed by repair provide a basis of debugging faults in code (assuming the erroneous states are due to bugs and not external events, say cosmic radiation). The key challenge to embodying this insight into a mechanical technique arose due to the difference in the concrete level where the program states exist and the abstract level where the program code exists: repair actions apply to concrete data structures that exist at run-time and have a dynamic structure (i.e., may get mutated), whereas debugging applies to code that has a static structure. I devised a lightweight approach to address this problem by abstracting repair actions, and using dataflow and control-flow guided heuristics to synthesize a corrective program transformation. Experimental results show that it was highly effective in fixing bugs in programs manipulating textbook data structures as well as programs from SpecJVM and DaCapo benchmarks.

An interesting use of techniques developed for program repair is to make data structure repair more efficient [6]. Data structure repair allows software to recover from errors. It is indispensable in scenarios such as hard disk crashes where a file system needs to be recovered. Data structure repair uses an expensive backtracking algorithm for systematic state transformation to find corrective changes. Our observation was that

many hardware and software faults are repetitive in nature, for example error occurs when a corrupted memory location is accessed, or a faulty method is invoked. We implemented a scheme to memoize the repair actions performed by data structure repair routine as repair templates, independent of any code and focused at fixing a known bug. This enables efficient data structure repair as the repair templates are tried before the expensive search is performed.

I also explored SAT based automatic program repair [7]. The key insight for this work was to replace a faulty statement that has deterministic behavior with one that has nondeterministic behavior, and use the specification constraints to prune the ensuing nondeterminism and repair the faulty statement. We used Alloy as an enabling technology for specifying and constraint solving. SAT based repair can handle not only structural errors but also behavioral violations. However, it requires prior localization of error and behavioral specifications.

Invariant Generation

Specifications enable a range of powerful software analyses, such as model checking, software testing, run-time verification, and data structure repair. However, manually writing a specification typically represents a significant burden on the developer, especially when specifying properties of programs that manipulate structurally complex data. Traditionally, developers rely on testing to check partial correctness. It is easier to provide test-inputs along with desired test-outputs that can be asserted to check validity. Daikon [8], a tool developed by Michael Ernst used tests to generate likely program invariants, however, at that time it could not detect properties of data structures. Using Daikon as an inspiration, we developed the first tool to generate likely invariants of complex data structures [9, 3]. Our main contribution was to view the program heap as an edge-labeled graph whose nodes represented objects and whose edges represented fields. We focused on generating graph properties, which included reachability. Our tool was able to discover interesting properties of various textbook data structures.

Matrices provide a natural representation for graphs. In spectral graph theory many properties of graphs are detected using spectral factorization of their representational matrices. I investigated if properties of program heaps can be discovered using the same technique — and the results were promising [10, 5].

Interdisciplinary Research

As a curiosity driven researcher, I have worked on some interesting projects over time. This has given me a broader view on solving complex problems and honed my abilities to work in an interdisciplinary research environment.

Probabilistic Modeling of Identity Attributes

After having completed core research work towards my PhD, I worked at the Center for Identity for a year and developed a probabilistic model of identity attributes. Losses caused by personal identity theft in the U.S. are staggering. Over 16 million people reported identity theft incidents in 2012 with a total financial loss exceeding $25 billion. Identity thieves exploit known methods for deriving data points from information exposed on the web as well as traditional approaches including stealing snail mail and making fake calls to ask for personal information. These gathered bits of information can be combined to reveal an alarming range of detail about a person's identity. We utilized reported cases of frauds in the news, and data gathered from law enforcement agencies to build a Bayesian Belief Network of identity attributes and their relationships. Given the evidence about some of the attributes, we were able to predict the attributes at higher risk or explain the likely point of entry by the identity thief. We also provided 3D visualization of the network for better human comprehension.

Object Based Video Segmentation

For my first Master's thesis in 2003, I worked on video segmentation. It is related to but distinct from image segmentation, as video segmentation requires consistency in assignment of pixels to objects across video frames. The goal of my research was to explore the use of robust statistics in a maximum a posteriori probability (MAP) framework that used multiple cues, like spatial location, color and motion, for segmentation. Our approach matched the performance of state of the art techniques at that time.

Future Work

I am interested in practical applications of my research and want my work to impact a large group of users in a meaningful way. I want to leverage the information found in software repositories and use it to improve automated debugging and enhance security. I am also interested in domain specific application of the techniques I have learnt and developed.

Noisy Channel Modeling of Programming Errors

The noisy channel model is a highly successful probabilistic framework, historically used in spell checkers and machine translation systems. Its goal is to find the intended word where the letters of a given word have been scrambled. An important property of words is that they come from a dictionary and their probabilities can be computed from large corpora of text composed of books, articles and news-feeds. I have observed that many of the programming errors arise from a developer's poor understanding of language constructs or incorrect use of programming idioms. These errors occur even when the developer has a good understanding of the algorithmic steps in his/her program and fixing them can be time consuming. I want to use software repositories to build an a priori model of such errors and use them to compute the best posterior action required for correction using the evidence of error. However, programs do not come from a dictionary and the technique cannot be directly applied. I hypothesize that an intermediate language that can capture "programming idioms" as words in a dictionary can help overcome this technical challenge. Such a system will be highly useful in providing feedback to programmers as well as those who are learning to program.

Code Diversification

The challenge of code diversification is to find functionally equivalent code but with different code structure and program state. It has been observed that many security threats exploit the static structure of the code to learn and exploit a vulnerability once, and then target other software that use the same code. A well known example of this is the Microsoft Patch-Tuesdays followed by reverse-engineered hacks on Exploit-Wednesdays. A simple multi-compiler attempts to solve this problem by changing the return addresses of functions. I have observed during my work on automated debugging that many times the repair routine found unexpected but equivalent programs. I think that building on this work will produce much richer equivalent programs that will be more impervious to vulnerabilities.

Model Checking of Cyber-Physical Systems

Computers controlling physical systems pose a major challenge for reliability. The critical infrastructure these systems control such as cars, aeroplanes and medical devices require strong guarantees of their correctness. Simulation using models (LabVIEW, StateFlow or SimuLink) is traditionally used in the industry to validate these systems. However, identifying hard-to-find design errors requires the use of formal methods. This presents a direct opportunity for applying my work done in generating representation invariants of data structures. LabVIEW and SimuLink models can be abstracted to directed acyclic graphs. Discovering invariants of this abstract representation can help prove various safety properties of the underlying models. Violations of design properties as well as basic errors such as dead-logic can be detected in this manner.

Long Term Goals

I want to make programming more accessible. As computing pervades our everyday lives and Internet of things becomes a reality, we will have to engage many more programmers and make the end users smarter. Ability to devise plans and think logically is a basic human trait. However, majority are not trained to translate their logic to efficient algorithm and reliable imperative code. I want to enhance the programming tools such that an average person can produce high quality software. I am excited and very hopeful given the promising advances in program synthesis, test case generation, automatic error correction and a host of enabling technologies.

References

[1] Muhammad Zubair Malik, Khalid Ghori, Bassem Elkarablieh, and Sarfraz Khurshid. A case for automated debugging using data structure repair. In *ASE 2009, 24th IEEE/ACM International Conference on Automated Software Engineering, Auckland, New Zealand, November 16-20, 2009*, pages 620–624.

[2] Muhammad Zubair Malik, Junaid Haroon Siddiqui, and Sarfraz Khurshid. Constraint-based program debugging using data structure repair. In *IEEE Fourth International Conference on Software Testing, Verification and Validation, ICST 2011, Berlin, Germany, 21-25 March 2011*, pages 190–199.

[3] Muhammad Zubair Malik, Aman Pervaiz, and Sarfraz Khurshid. Generating representation invariants of structurally complex data. In *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, pages 34–49, 2007.

[4] Muhammad Zubair Malik. Dynamic shape analysis of program heap using graph spectra. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, pages 952–955.

[5] Muhammad Zubair Malik and Sarfraz Khurshid. Dynamic shape analysis using spectral graph properties. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, Montreal, QC, Canada, April 17-21, 2012*, pages 211–220.

[6] Razieh Nokhbeh Zaeem, Muhammad Zubair Malik, and Sarfraz Khurshid. Repair abstractions for more efficient data structure repair. In *Runtime Verification - 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings*, pages 235–250.

[7] Divya Gopinath, Muhammad Zubair Malik, and Sarfraz Khurshid. Specification-based program repair using SAT. In *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 173–188.

[8] Michael D. Ernst. *Dynamically Discovering Likely Program Invariants*. Ph.D., University of Washington Department of Computer Science and Engineering, Seattle, Washington, August 2000.

[9] Sarfraz Khurshid, Muhammad Zubair Malik, and Engin Uzuncaova. An automated approach for writing alloy specifications using instances. In *Leveraging Applications of Formal Methods, Second International Symposium, ISoLA 2006, Paphos, Cyprus, 15-19 November 2006*, pages 449–457.

[10] Muhammad Zubair Malik, Aman Pervaiz, Engin Uzuncaova, and Sarfraz Khurshid. Deryaft: a tool for generating representation invariants of structurally complex data. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 859–862.