RESEARCH STATEMENT

Miryung Kim
Assistant Professor of Electrical and Computer Engineering at UT Austin
Reporting Period: January 2009 – August 2014

Over the last several decades, software systems have benefited and permeated every aspect of our lives. Continuing this advancement will require not only creating new innovative software but also improving the ability to evolve existing software systems. Given that 90% of the cost of a typical software system is incurred during the maintenance phase (Madhavji et al. 2006), improving the capability to evolve software will drastically improve productivity and competitiveness of the software industry.

My mission is to improve developer productivity and software correctness in evolving large software systems. With a primary focus on evolution, my students and I design, implement, and evaluate automated analysis algorithms and development tools that make code changes easy to reason about, reusable, and portable. I also conduct user studies with professional software engineers and carry out statistical analysis of version histories to allow data-driven decisions for designing novel tools.

A majority of my research effort focuses on explicitly supporting systematic changes. This research theme is based on the insight that high-level software modifications are often systematic, requiring similar but not identical changes to similar contexts. My doctoral research investigated why and how systematic changes occur through a study of code duplication [FSE ’05 and ISESE ’04] and introduced a rule-based program differencing approach to summarize systematic changes [ICSE ’07 and ICSE ’09]. Since coming to UT Austin, my students and I developed various innovative approaches to automate systematic changes, to empirically study the extent and characteristics of systematic changes, and to reconstruct systematic changes from program versions.

The research problems that I address are long-standing, foundational problems in software engineering. Through empirical studies, I bring new insights to these problems and develop innovative solutions for improving developer productivity.

RESEARCH CONTRIBUTION HIGHLIGHTS, VISIBILITY, AND RECOGNITION

Since joining UT Austin, I successfully ramped up a strong research group, which published 9 papers at ACM/IEEE International Conference on Software Engineering (ICSE), ACM International Symposium on Foundations of Software Engineering (FSE), ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), and ACM SIGPLAN conference on Systems, Programming, Languages and Applications: Software for Humanity (OOPSLA) between 2009 and 2013—ICSE and FSE are top-tier software engineering conferences with acceptance rates below 15% and PLDI and OOPSLA are top-tier programming language conferences. I received an IBM Jazz Innovation Award in 2009, a Microsoft Software Engineering Innovation Foundation Award in 2011, and a Google Faculty Award in 2014. I was invited to visit Microsoft Research in 2011, and I quantified the impact of Windows 7 refactoring through statistical analysis of version histories. The awards and invitation attest to the software industry’s appreciation of the impact and potential of my research.

Along with my former students, Adam Duley and Chris Spandikow, I received an ACM SIGSOFT Distinguished Paper Award at the 25th IEEE/ACM International Conference on Automated Software Engineering for our paper “A Program Differencing Algorithm for Verilog HDL” (top 1.5% of 190 submissions). This paper presented a novel program-differencing algorithm by applying software evolution analysis to a new hardware design domain. In many software engineering conferences, top reviewed papers are selected by the program committees and invited for journal publications. Over the past five years, three of my papers were invited for journal publications.
According to Google Scholar citations, my research on code duplication sparked new interest in code cloning research. My work on program differencing brought new attention to multi-version program analysis and inspired many approaches that make meaningful changes explicit and suppress non-essential changes. My research is supported by competitive, peer-reviewed funding sources, including an NSF CAREER award.

In the following, I present details of selected research projects. The cited work, along with a complete list of co-authors can be found in my CV and also at http://users.ece.utexas.edu/~miryung

**Automating Repetitive Changes using Examples**

Maintaining multiple similar versions of software and making similar changes is error prone and time consuming. For example, when libraries and frameworks change APIs, all client applications must update their outdated API usage code. As another example, developers also manually port patches from peer projects, such as FreeBSD and NetBSD. Automation of systematic changes can reduce tedious programming work and errors. Existing approaches are inadequate—refactoring engines support only predefined, semantics-preserving transformations. Source transformation tools require developers to prescribe edits in advance and use a formal syntax.

The PhD research of my student Na Meng, presents novel algorithms to automate systematic changes. Our first algorithm, SYDIT, learns an abstract context-aware program transformation script from a change example that is specified by a developer and applies it to a user-selected target [PLDI ’10]. Our key insight is that by learning abstraction transformation from examples, we can automate systematic edits in a flexible and easy-to-use manner. In the real world application of SYDIT, we have shown that SYDIT produces code that is 96% similar to the manual editing done by programmers.

This SYDIT project led to the development of LASE, which finds change locations automatically and applies systematic program edits [ICSE ’13]. Our insight is that by avoiding over-generalization and over-specialization through the use of multiple examples, we can correctly infer the edit context, which then serves the dual purposes of finding change locations and applying customized edits. In our evaluation of real world bug fixes, LASE was able to apply bug fixes to the locations missed by human developers, correcting errors of omissions. These errors were later confirmed by developers. This method shows the potential of repairing software by using version histories.

**Analytical Support for Investigating Software Modifications**

Software engineers often need to analyze software modifications during code reviews and bug investigations. My students and I have invented a suite of code change analyses and tools that can help programmers investigate code modifications and reduce bugs.

- **Analysis of Cross-System Porting.** A recent study finds a general lack of uniqueness in software, which indicates a significant level of redundancy across different applications. Software forking (creating a variant product by copying an existing project) contributes to this problem and causes repeated work across different projects. Our recent study of NetBSD, OpenBSD, and FreeBSD finds that 10 to 15% of all changes in the BSD family are actually recurring patches that are ported from peer projects [FSE ’12]. We also investigate the types of porting errors found in practice and how to automatically detect and characterize potential porting inconsistencies [ASE ’13].

- **Rule-based Program Differencing.** Our insight is that based on their common structural characteristics, low-level changes can be summarized concisely by using a formalism of first order logic rules. I invented a rule-based differencing approach that discovers and represents systematic changes as logic rules [ICSE ’09, TSE ’13]. This approach has taken hold in the research community, as evidenced by many approaches that summarize meaningful changes and suppress non-essential changes.

- **A Logic-Query Approach to Refactoring Reconstruction.** Our insight is that the skeleton of refactoring edits can be expressed as a logical constraint about a program structure before and after refactoring. RefFinder automatically discovers the types and locations of refactoring edits using a logic query approach. Unlike existing approaches, it is highly extensible, because supporting a new refactoring type requires simply adding a new rule [ICSM ’10].
• **FaultTracer Change Impact Analysis.** Our insight is that by applying spectrum-based fault localization in tandem with an enhanced change impact analysis, it could identify and rank failure-inducing edits accurately. In the real-world application, FaultTracer significantly reduces developers’ effort in identifying the causes of regression failures [ICSM ’11].

• **Program Differencing for Hardware Description Languages.** Our insight is that a differencing algorithm must account for concurrent computation, which is common in hardware description languages. With my students, who are hardware design engineers at AMD and IBM, I invented a position-independent differencing algorithm [ASE ‘10, ACM SIGSOFT Distinguished Paper Award, ASE Journal ’12]. Our user study with hardware design experts at a large semiconductor company shows that the experts’ change classification is better aligned with Vdiff than diff. “Vdiff ignores things that you don’t really care about.” “It is great for team leads to understand what has changed without having to look at the textual differences.”

**Assessing the Impact of Refactoring during Software Evolution**

It is widely believed that refactoring improves software quality and productivity by making it easier to maintain and understand software systems. EXtreme Programming claims that a lack of refactoring incurs technical debt and advocates the rule of refactor mercilessly. On the other hand, unlike new features or bug fixes, some believe that refactoring does not provide immediate benefit. To create a scientific foundation that can help engineers to make data-driven refactoring decisions, we investigated the role of refactoring during software evolution through statistical analysis of version history data and user studies.

We first investigated the relationships between API-level refactors and bug fixes by using a fine-grained version history of open source projects. Using our API refactoring reconstruction algorithm, we found that the location and timing of API refactoring is strongly correlated with bugs [ICSE ’11].

Building on this effort, I was invited to visit Microsoft Research. I quantified the impact of a multi-year Windows 7 re-architecting effort [FSE ’12]. This project was particularly important within Microsoft, because despite the multi-year refactoring investment, there had not been a systematic quantitative validation of the refactoring impact on productivity and quality measurements. In collaboration with Tom Zimmermann and Nachi Nagappan, I analyzed version history data, conducted a survey of over three hundred developers, and interviewed the architects and development leads to assess the impact of refactoring. My study found that refactoring contributed to the reduction of inter-module dependencies and post-release defects, which showed the tangible benefit of refactoring. We are continuing our investigation into the impact of refactoring on size, churn, complexity, test coverage, failure, and people and organization measures [TSE ’14]. At the end of my visit, I was invited to present the research outcome to the entire Windows kernel development organization. This is a rare opportunity for an academic visitor and is another piece of evidence that shows the direct impact of my research to software industry. This research has demonstrated the potential to change the management of the multi-billion dollar software industry by providing a sound basis for assessing the impact of refactoring.

**SOFTWARE TOOLS AND DATA sets**

I believe in presenting innovative software analysis algorithms in a manner that is approachable to and usable by professional developers. My group has demonstrated our software tools at premier venues for formal research demonstrations [ICSE ’10, FSE ’10, FSE ’11, FSE ’12, FSE ’12, ICSE ’13, ICSE ’14]. We have also made our analysis artifacts and scripts available for further studies. Our tools and research data are available at [http://user.ece.utexas.edu/~miryung](http://user.ece.utexas.edu/~miryung).