

Research Statement

Lingxiao JIANG

School of Information Systems, Singapore Management University

Tel: (65) 6808-5113; Email: lxjiang@smu.edu.sg

8th February 2012

Background

Software systems are becoming more and more sophisticated, complex, and ubiquitous. It is highly critical to increase development productivity, ensure high-quality software, and reduce development and maintenance cost. Past studies in US have estimated that software maintenance and license fees accounted for \$86 billion or 41% of the \$210 billion in revenue collected by software vendors in 2005, and that software errors annually cost \$59.5 billion, and that debugging costs a typical large organization between 5.2 to 22 million US dollars in 2008.

With the growth and evolution of software systems in the past decades, it is becoming clear that much valuable knowledge must have been embedded in various parts of various software systems, and should be analyzed and reused for software maintenance and new development. With the accumulation of large amount software data, not only the source code itself, but also much contextual information, such as code change histories, test suites, execution profiles, bug databases, email exchanges among developers, and code design documents, analyzing such contextual software data has become a promising way to learn programming patterns, API usages, design experiences, bug signatures, reusable programming knowledge, etc. that can help to improve software quality and reduce development and maintenance costs.

Research Areas

My main research goal is to develop techniques, tools, and methodologies that can help to speed up development, reduce bugs, facilitate maintenance tasks, and thus save developer time and cost.

A general theme of my work is software mining and analysis, especially context-aware code search and analysis, where rich contextual information beyond source code itself is utilized to analyze the properties of software and extract reusable programming knowledge. My work aims to design and implement new software data analysis techniques, and extract prior knowledge expressed in various forms, such as directly reusable code fragments, programming patterns, API specifications, and bug signatures, and automate certain development and maintenance tasks with the reusable knowledge, such as code refactoring, bug detection, fault localization, and program synthesis.

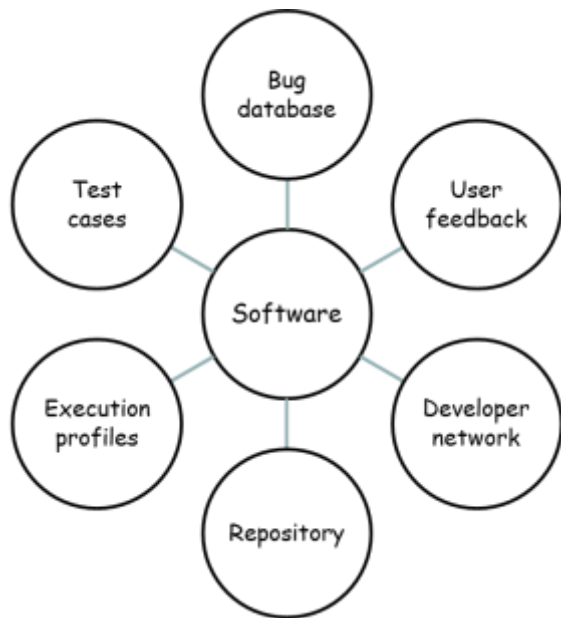


Figure 1. Contextual Software Data

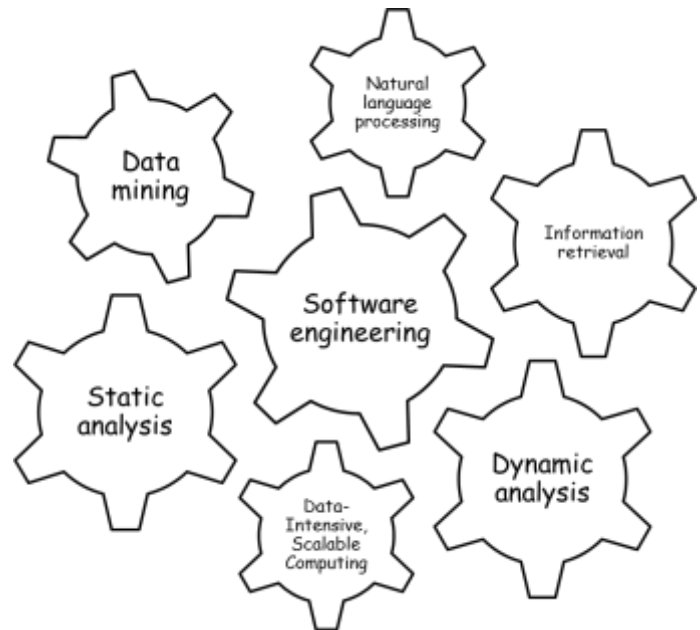


Figure 2. Software Mining & Analysis Techniques

To enable efficient analysis of various data sources (Figure 1) and the extraction of reusable knowledge, my work designs and implements efficient search and analysis techniques that combine techniques from multi-areas, including static & dynamic program analysis, software engineering methodologies, data mining, information retrieval, natural language processing, and distributed computing techniques (Figure 2).

Scalable Context-Aware Code Search

During development and maintenance processes, programmers need to either write new code or modify existing code. How often do they stop and look for some code written before, trying to understand and reuse that code? Could we provide automated ways to facilitate code search and reuse for programmers without major interruptions to their development and maintenance activities?

One aspect of my research in code search is to detect and analyze similar code fragments (a.k.a. code clones) in large programs. Our studies and others' have noticed that large programs often contain more than 20% cloned code, which often leads to unnecessarily redundant maintenance cost and more subtle software defects that should be avoided. Detecting code clones, tracking their migration and evolution among large programs, and managing them properly have many important applications, such as code refactoring, optimization, and bug detection. My research proposes new, general detection techniques, based on both program syntax and semantics (e.g., parse trees, dependency graphs, and functional behaviors), that can scalably and accurately detect code clones and enable the mentioned applications on million-line programs.

Another aspect of my research in code search is to efficiently retrieve code intended by developers (a.k.a. code queries) so as to help developers locate useful code quickly. How can developers easily express their code query intentions? How can high level intentions which may be described in natural languages be linked to source code? How can complex code structures and even functionality be searched quickly to match the intentions? My research combines techniques from program analysis, information retrieval, data mining, and natural language processing to tackle the problems.

Scalable and Accurate Tree-based Clone Detection. We have developed Deckard, a tree-based clone detection technique and tool [1,14], based on the insight that complex code structure similarity problems can be reduced to easier vector similarity problems. Our experiments have shown that Deckard scales to millions of lines of code in various programming languages with few false positives.

Graph-based Scalable Detection of Clones. The same key insight proposed in Deckard is also applicable to other structures than trees, such as program dependency graphs. Since program dependency graphs capture code semantics to some extent, we are thus able to detect semantic-aware code clones by reducing NP-hard graph matching problems to near linear vector similarity problems [11].

Functionally Equivalent Code Mining. Our work also pushes semantic clone detection to a different level by developing EqMiner, a practical approach for mining functionally equivalent code fragments [9]. It adapts automated random testing and eagerly partitions arbitrary code fragments into functionally equivalent clusters. Evaluated on the Linux kernel, it discovered many functionally equivalent, but syntactically different code fragments, and motivates future research on functionality-aware code refactoring and reuse.

Context-Based Inconsistencies for Detecting Clone-related Bugs. We also observed that inconsistent code changes among code clones often lead to bugs. Thus, we introduced a general notion of context-based inconsistencies to detect clone-related bugs [13]. Many previously unknown bugs of diverse characteristics in large projects, such as the Linux kernel, Eclipse, and a well-known product from a well-known software company, were discovered this way. Further, we utilize active user feedbacks and a nearest neighbor classification technique to iteratively refine the bug reports and make them significantly more accurate [2].

Topic Modeling for Code Search. In addition to clone detection and analysis, we aim to use rich data beside code itself to make code search more accurate. One kind of rich data we are using is documents about the code design, feature descriptions, and code comments for a project, which are written in natural languages. Our work employs topic modeling techniques on both such rich data and source code itself to establish linkage between them [4], and also enrich the traditional code representations (syntax trees, call graphs, dependence graphs, etc) with semantic topic labels, and enable more accurate code query based on topics decide the topic labels [5].

Future Research Directions. With the advances of code search and analysis techniques and the accumulation of large number of software projects, a whole new way of code refactoring and reuse is becoming possible: Given a large library of commonly used code fragments and related programming knowledge (such as API usage patterns) and efficient, accurate code search capability, developers may just need to know how to compose existing code so that it fits for a new context, instead of writing much new code. A number of research questions I am interested in and believe they will be helpful for facilitate software development and maintenance:

- What would be the organization of such a large code library so that search of and access to any code fragment and sample in the library in terms of syntactic and semantic similarity is efficient? Curated database may not be a good choice due to its inflexibility; multi-indexing and tagging of code at various granularities could be a better choice.
- What would be the interface to represent a reusable code fragment so that it is easy for developers to access, understand, and apply, and facilitate code reuse and composition? We may need a query interface that could accept various kinds of code queries, and a code adapter that could change code search results to fit the context used for the query.
- For particular domains, such as mobile applications, web applications, and enterprise information systems, what may be the domain-specific characteristics that could be exploited to facilitate adoption of code search and analysis techniques in the domains? Various programming frameworks may change the way developers write programs; additional patterns in a domain could be better analyzed by customized approaches, even though general code search and analysis can be applied directly.

Context-Aware Automated Testing and Debugging

I also believe that rich software data besides source code, such as history of code changes, execution profiles from users, and socio-technical information stored in software repositories that relates to the complex interactions between people and technology in software development processes, can be utilized to improve software development and maintenance processes. In particular, I am interested in how such data can be used to improve the effectiveness of testing and debugging, which are often labor-intensive and account for more than 50% of the cost in software development.

Profile-Based Fault Localization. The idea is that we could accumulate large amount of information about a program from in-house test cases or many users in the field. Even if the information from each user is sparse, we could infer many properties about the program without compromising user privacy and performance. We have developed an approach that uses machine learning algorithms on accumulated execution profiles to predict bug locations, and uses static program analysis to construct control flow paths that may reveal the cause-failure transition paths [12]. We also realized that no single fault localization technique

outperforms all others, and have investigated the effectiveness of more than 20 association measures on the fault localization problem [8], and employed multiple search algorithms to construct a composite fault localization technique that can perform better than all previous fault localization techniques [3].

Future Research Directions. With various software data in-hand (Figure 1), I feel the research would become more and more interdisciplinary (Figure 2): although program analysis may still be the major technique for analyzing code itself, solving many software engineering problems would draw techniques from different areas, such as data mining, natural language processing, information system management, intelligent systems, and even from areas beyond computer science, such as social science and economics. I am very interested in making progresses in these directions:

- Scalable context-aware analysis: No previous study has analyzed all of the rich data surrounding a software project together. Working with my colleagues, we will take a holistic view, aggregating all of the available data, including source code, repository data, and socio-technical networks among developers, combining traditional program analysis with data mining, natural language processing, and information visualization techniques, and aim to provide rich contextual information for any single piece of data and facilitate developers to understand the code history and context when debugging.
- Scalable testing and debugging for large systems: When a program becomes larger or when many software components are integrated together, it becomes challenging to just identify a failure-causing component, not to mention the need to identify the root cause within a component. It would be of high impact if we could scale up the automated testing and debugging techniques we have developed for single program for large systems. On one hand, we would need to design new mechanisms for analyzing large systems, and likely to take domain knowledge specific to the systems for more scalable analysis. On the other hand, we would also pursue approaches that may abstract or simplify the systems themselves, as we tried before using profiles [10].
- Privacy-aware testing and debugging: When we take a holistic way to analyze software data, including test inputs, privacy concerns would become a serious issue. For example, if a health-care system contains a lot of patient records which should only be viewable by authorized medical staff, how could we still utilize such data for software mining and analysis without compromising patient privacy? We have started to investigate how program analysis and database and data mining techniques can be combined to address the issue [7].

I will also remember that the ultimate success of these studies would rely on how well the developed techniques, tools, and methodologies can be actually incorporated into the many different phases of software engineering processes. Paying attention to the advances and needs of the software industry and keeping a strong connection with the real world would be a critical factor that decides the value and impact of my research.

Selected Publications and Outputs

- [1] Lingxiao Jiang. DECKARD: A Tree-Based Code Clone and Clone-Related Bug Detection Tool. Open source package for download: <https://github.com/skyhover/Deckard>.
- [2] Lucia, David Lo, and Lingxiao Jiang. Active Refinement of Clone Anomaly Reports. To appear in the proceedings of the 34th International Conference on Software Engineering (ICSE'12), Zurich, Switzerland, June 2-9, 2012. ACM/IEEE.
- [3] Shaowei Wang, David Lo, Lingxiao Jiang, Lucia, and Hoong Chuin Lau. Search-Based Fault Localization. In the proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11), Lawrence, Kansas, USA, November 6-12, 2011. IEEE.
- [4] Shaowei Wang, David Lo, Zhenchang Xing, and Lingxiao Jiang. Concern Localization using Information Retrieval: An Empirical Study on Linux Kernel. In the proceedings of the 18th Working Conference on Reverse Engineering (WCRE'11), pages 92-96, Limerick, Ireland, October 17-20, 2011. IEEE Computer Society.
- [5] Shaowei Wang, David Lo, and Lingxiao Jiang. Code Search via Topic-Enriched Dependency Graph Matching. In the proceedings of the 18th Working Conference on Reverse Engineering (WCRE'11), pages 119-123, Limerick, Ireland, October 17-20, 2011. IEEE Computer Society.
- [6] Aditya Budi, Lucia, David Lo, Lingxiao Jiang, and Shaowei Wang. Automated Detection of Likely Design Flaws in N-Tier Architectures. In the proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'11), pages 613-618, Miami Beach, Florida, USA, July 7-9, 2011. Knowledge Systems Institute Graduate School.
- [7] Aditya Budi, David Lo, Lingxiao Jiang, and Lucia. kb-Anonymity: A Model for Anonymized Behavior-Preserving Test and Debugging Data. In the proceedings of the 32nd ACM SIGPLAN Programming Language Design and Implementation (PLDI'11), pages, 447-457, San Jose, California, USA, June 4-8, 2011. ACM.
- [8] Lucia, David Lo, Lingxiao Jiang, and Aditya Budi. Comprehensive Evaluation of Association Measures for Fault Localization. In the proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM'10), Timisoara, Romania, September 12-18, 2010. IEEE Computer Society.
- [9] Lingxiao Jiang and Zhendong Su. Automatic mining of functionally equivalent code fragments via random testing. In the proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA'09), pages 81-92, Chicago, Illinois, USA, July 19-23, 2009. ACM.
- [10] Lingxiao Jiang and Zhendong Su. Profile-guided program simplification for effective testing and analysis. In the proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'08), pages 48-58, Atlanta, Georgia, USA, November 9-14, 2008. ACM.
- [11] Make Gabel, Lingxiao Jiang, and Zhendong Su. Scalable detection of semantic clones. In the proceedings of the 30th International Conference on Software Engineering (ICSE'08), pages 321-330, Leipzig, Germany, May 10-18, 2008. ACM.
- [12] Lingxiao Jiang and Zhendong Su. Context-aware statistical debugging: From bug predictors to faulty control flow paths. In the proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07), pages 184-193, Atlanta, Georgia, USA, November 5-9, 2007. ACM.
- [13] Lingxiao Jiang, Zhendong Su, and Edwin Chiu. Context-based detection of clone-related bugs. In the proceedings of the 6th joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE'07), pages 55-64, Dubrovnik, Croatia, September 3-7, 2007. ACM.
- [14] Lingxiao Jiang, Ghassan Mishserghi, Zhendong Su, and Stéphane Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In the proceedings of the 29th International Conference on Software Engineering (ICSE'07), pages 96-105, Minneapolis, Minnesota, USA, May 20-26, 2007. IEEE Computer Society.