Research Statement

Lingxiao JIANG
School of Computing and Information Systems, Singapore Management University
Tel: (65) 6808-5113; Email: lxjiang@smu.edu.sg
27 December 2023

Background, Research Motivation, and Central Concerns

It is important for programmers to have high productivity and ensure code quality during software development and maintenance. Some labor-intensive activities, such as understanding the meaning of existing code, analyzing the properties and behaviors of the code, and identifying relevant code that needs changes, are very commonly performed by programmers during development and maintenance. Past studies have shown that programmers spend 20% to 80% of their time in code surfing to search, analyze, relate, and understand the code in large complex software systems before they are able to change the code to adjust functionalities or fix bugs or vulnerabilities. Such code surfing activities are not limited to browsing through code, it may also involve performing code testing and analyzing code execution logs, searching additional documents in the internet, and participating in online discussions. As manpower is a major component in the cost of software development and maintenance, saving just 5% of programmers' time for more than 20 million programmers in the world by speeding up code search, analysis, comprehension, and change processes could lead to an improvement in productivity worth millions and billions of dollars a year globally. With improved developer productivity and increased degree of automation during development and maintenance, software quality can be expected to improve too and potentially save billions of dollars annually caused by software errors.

My main research goal is to develop new methodologies, techniques, and tools that can help programmers to search, identify, analyze, understand, and change the code they need and thus help to increase productivity, reduce maintenance costs, and improve software quality. My work explores new solutions for software engineering problems related to various aspects of code search, comprehension, analysis, testing, debugging, repair, refactoring, translation and synthesis, and contributes to the state-of-the-art in the literature. On one hand, I aim to address some fundamental and inherent challenges faced by these problems, including the "program representation" challenge that can construct efficient and effective representations capturing various aspects of properties about software code for code search and analysis, and the "scalability" challenge that can enable internet-scale code search and analysis. On the other hand, I aim to understand and address practical problems faced by programmers in real world via various kinds of "empirical studies", and provide useful and usable solutions, such as suggesting reusable code for their coding tasks, predicting where buggy code may be for faster debugging, and generating functioning code that fixes the bugs. Developing new techniques enhancing traditional software mining and program analyses in combination with recent progress in deep code learning are important aspects of my research to address the challenges.

My research theme is also enriched by and benefited from the growth, evolution, and openness of software projects in the past decades, where more and more socio-technical data in and around software projects (e.g., code itself, design documents, code change histories, bug reports, discussions among developers, and feedback from users) has been accumulated. Together with open-source movements in the past decades, it is evident that much valuable knowledge has been embedded in various parts of diverse software systems, and should be analyzed and reused for software maintenance and development. Learning and analyzing contextual data together with code has become a promising way to learn various programming knowledge (e.g., design patterns, reusable code, bug fixes, common refactoring operations, etc.) that can help to improve developer productivity, software quality and reduce costs. Therefore, my work aims to enable "multi-modal code search and analysis" that can utilize rich contextual information and be versatile in extracting various kinds of reusable patterns and knowledge for different programmers' needs under different situations.

Research Areas

Figure 1 illustrates many areas in the field of software engineering that involves software design, development, deployment in both the forward direction and the backward reverse engineering/reengineering.

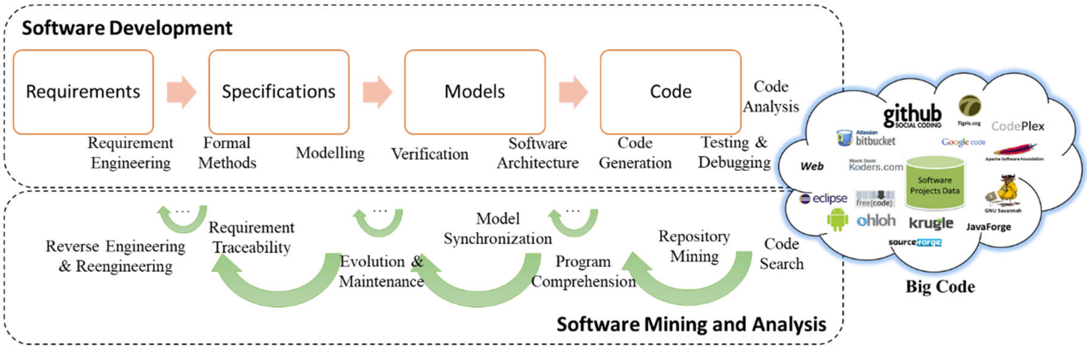
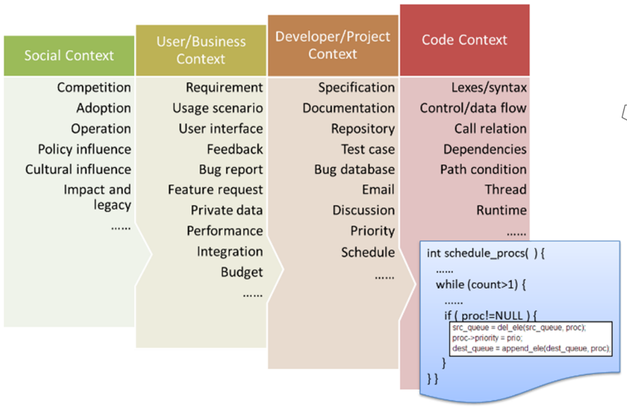


Figure 1: Software Mining and Analysis within "Bidirectional" Software Engineering

My work is positioned near to the right end in this "bidirectional software engineering" field. It is to analyze existing big code bases involving diverse kinds of software data, to locate and extract reusable patterns and knowledge, and thus to help programmers to ensure code quality and improve productivity. Particularly, I work on code search and repository mining to facilitate program comprehension by programmers, and on code analysis, testing, and debugging to facilitate software quality assurance.

The kinds of software data used as my research subjects range from source code to diverse kinds of code contexts. As illustrated in Figure 2, there are many kinds of contexts surrounding a given piece of code, from its lexical and syntactic structures to test cases and execution profiles, from its developers' interaction to users' feedback, from its performance to operation environments, from its functionalities to social impact and legacy. My work utilizes many contexts in the two columns towards the right side, and some contexts in the columns on the left. Note that diverse and comprehensive kinds of socio-technical information related to the code are important and useful for helping programmers to search, understand, analyze, and change code. In the future, I plan to expand further to the left side to improve the techniques and tools I have developed and make my work more practically and socially useful.

More technically, my work ranges from static and dynamic program analysis to many kinds of heuristic and statistical analyses, such as deep learning, data mining, information retrieval, search heuristics, and empirical studies. Integration of a variety of techniques has benefited my work to be able to handle diverse kinds of socio-technical information, as illustrated in Figure 3. In the future, I will continue to expand and strengthen my interdisciplinary work.



Social Context	User/Business Context	Developer/Project Context	Code Context
Competition	Requirement	Specification	Lexes/syntax
Adoption	Usage scenario	Documentation	Control/data flow
Operation	User interface	Repository	Call relation
Policy influence	Feedback	Test case	Dependencies
Cultural influence	Bug report	Bug database	Path condition
Impact and legacy	Feature request	Email	Thread
……	Private data	Discussion	Runtime
	Performance	Priority	……
	Integration	Schedule	
	Budget	……	
	……		

```
int schedule_procs( ) {
  ……
  while (count>1) {
    ……
    if ( proc!=NULL ) {
      src_queue = del_ele(src_queue, proc);
      proc->priority = prio;
      dest_queue = append_ele(dest_queue, proc);
    }
} }
```

Figure 2: Contexts for Software Mining and Analysis

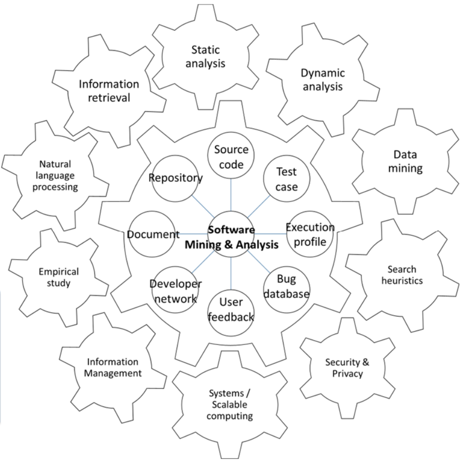


Figure 3: Interdisciplinary topics

As such, a general theme of my work is software mining and analysis, especially context-aware, multi-modal code search, analysis, testing and debugging, where contexts can be diverse kinds of information beyond source code itself. I design and implement new techniques and tools for extracting reusable programming knowledge in various forms, such as directly reusable code fragments, programming patterns, API specifications, bug signatures, and code refactoring rules, etc., so that the techniques and tools can facilitate various software engineering tasks, such as code refactoring, bug detection, fault localization, privacy-aware testing, and program synthesis, and help programmers to more efficiently search, understand, analyze, change code for implementing new functionalities or fixing bugs. I also keep broadening potential impact of my work by (1) adapting the code search, analysis, testing and debugging techniques to software development and maintenance activities in high-value, infrastructural software (e.g., the Linux kernel, the Android platform, the Ethereum virtual machine, etc.) and strategic domains related to SMU's Research Focus Areas in Digital Transformation (https://www.smu.edu.sg/research, e.g., mobile systems and application development and analysis, cybersecurity, financial systems, blockchain and smart contracts, AI systems), and (2) incorporating methodologies and techniques from information management research to analyze the impact of the techniques on people (including programmers, managers, and users) to form a feedback loop for improving the usability and usefulness of my work.

More specifically, my work can be classified into two main overlapping categories: code search & analysis, and testing & debugging, including various kinds of techniques adapted to address the problems and empirical studies carried out to understand the problems and techniques better.

Scalable Context-Aware Code Search & Analysis

During development and maintenance processes, developers need to write new code or modify existing code. How often do they stop and search for some code written before, and try to understand, reuse that code? Can we provide automated ways to facilitate code search, understanding, and reuse without major interruptions to developers' development and maintenance activities?

One aspect of my research 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. Sample work includes scalable syntactic and semantic clone detections [Tool/Deckard, ICSE2007, ICSE2008, ISSTA2009], clone-based detection of code inconsistency bugs [FSE2007, ICSE2012], and code refactorings [ISSTA2014, FSE2014].

Another aspect of my research is to efficiently retrieve code intended by developers (a.k.a. code queries) 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 functionality be searched quickly to match the intentions? Can we utilize the social interactions among developers to help improve search results? My research combines techniques from program analysis, information retrieval, data mining, and natural language processing to tackle the problems. Sample work includes topic-enriched code search [WCRE2011a], concern localization [WCRE2011b], socio-technical data for code search and analysis [ICSM2012, APSEC2012, SAC2013, CSMR2013a, CSMR2013b, HICSS2015], cross-language code search [IWSM2016, ASE2016, IEICEJournal2017], and API recommendation [Tool/LibraryGuru].

A fundamental aspect of my work on code search and analysis is to explore new ways to represent code that can capture essential meaning of the code and enable efficient and accurate search and analysis. Preliminary results have been obtained for certain code analysis tasks, such as algorithm classification and cross-language API mappings, by using combinations of program analysis and deep neural networks [ICSE2018NIER, SANER2019, FSE2019, AAAI2021, ICSE2021, SIGIR2021].

Future Research Directions.

When more diverse kinds of data related to projects accumulate from various application domains, there are new challenges facing software mining & analysis. We need to take an interdisciplinary approach (Figures 2 & 3) to take all such data into consideration and provide better results for users:

- How can we define a flexible notion of "clones" that can be tailored to developers with varying experience and applications of different purposes?
- Given data from heterogeneous sources of heterogeneous formats, such as source code repositories, discussion forums, online blogs, etc., how can we organize such raw data in a way best suitable for cross-data search?

- How can we design new software mining & analysis algorithms that can efficiently run across various raw data? Or would it be better to run different algorithms on different data and then combine results?
- When low-level code fragments and programming knowledge (such as API usage patterns) are extracted and accumulated, how can we organize and translate them into high-level concepts that can be easier for developers to understand and reuse?
- What would be the unified interface that can be expressive enough for developers to describe what they want and enable developers to search and analyze software across various sources and abstraction levels?
- How can we put search and analysis results into the developers' social contexts and facilitate them to understand, help, and learn from each other?
- Can we understand better how developers learn and interact with each other, and use such knowledge to guide the design and development of our techniques?
- For specific domains, such as mobile applications, web applications, smart contracts, 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?
- For mobile systems and applications and other domains, can we adapt the techniques to identify similarities and differences among various parts that may be involved in different architectures, semantics and functionalities? What kinds of programming patterns can we identify for Android apps to facilitate code reuse, refactoring, optimization, and security and privacy protection?

Context-Aware Automated Testing and Debugging

During development and maintenance processes, testing & debugging activities are also essential for ensuring software quality and reliability. Such activities are often labor-intensive and account for more than 50% of the cost in software development. A main theme in my research in this aspect is to utilize various kinds of data, including software code itself and rich contextual data about the code (e.g., history of code changes, execution profiles of the program, and socio-technical information that relates to the complex interactions between people and technology in software development processes) for better testing and debugging.

Testing & debugging may also be viewed as a special kind of code search, as it aims to search for potential bugs in code, although the techniques can be quite different. My sample work includes profile-based fault localization [ASE2007, FSE2008, ICSM2010, ASE2011, ASE2012b, JSEPJournal2014, ASEJournal2015], privacy-aware testing & debugging [PLDI2011, ASE2012a], investigation of community-scale uses of testing [ISSRE2013, QSIC2013], Android system and app analysis [ASE2016, ICPC2017, ASE2022, ASE2023], and more recently on faster and more capable symbolic execution techniques [Tool/FastKLEE, CCS2023] and various emerging applications, such as blockchain and smart contracts [Tool/SmartEmbed, Tool/MandoGuru, DSAA2022, FSE2022, ISSTA2023, MSR2023b, NDSS2023], internet of things [APSEC2020], Arduino programming [MSR2023a], and intelligent systems [ICSME2020].

Future Research Directions.

Testing & debugging has been traditionally focused on software itself. However, with various software data available in hand, I believe the research will become more and more interdisciplinary as well (Figures 2 & 3). Static and dynamic program analysis may still be the major techniques for this purpose, many techniques from different areas will be drawn up to help utilize various data available and improve testing & debugging. I am very interested in making progresses in adapting techniques from data mining, deep learning, natural language processing, information management, and even social science and economics, to address practical software engineering problems. At the same time, I aim to make the analysis techniques, especially the ones based on deep learning more easily understandable by developers [ASE2019, ICSE2021, SIGIR2021, DSAA2022, MSR2023b].

- How can we incorporate data beyond code, such as user feedback, developer experience, project team organization, etc., into the decision making for testing & debugging?
- How can we improve the scalability of traditional program analysis techniques with new data mining, deep learning, natural language processing, and information management & analysis techniques, especially for large enterprise systems?
- How can we design testing & debugging techniques that are general enough to handle many kinds of software errors and their variants? Or do we need different kinds of tailored techniques for different kinds of errors?
- How can we tailor general testing & debugging techniques for specific kinds of issues (e.g., security vulnerabilities of specific patterns, performance degradations and/or private leaks of mobile applications or smart contracts) that may be extracted from Context-Aware Code Search & Analysis?
- How can we incorporate domain knowledge or programming patterns specific to certain systems (e.g., event-driven architectures for mobile applications, web applications and blockchain virtual machines) into the testing & debugging processes to make the processes more efficient and effective for those systems?
- How can we protect software users' privacy while using their feedback and data for testing & debugging, which may involve information, risk, policy management issues?

Summary

I believe that social contexts and relevance are very important for the ultimate success of these software engineering studies. There could be many various directions to go with the studies, and I will keep exploring and seek the most impactful directions. I will keep focusing on in-depth research in the techniques themselves, not only in the techniques of my own core areas in traditional program analyses and software mining, but also in the techniques of many related areas for diverse, multi-modal contextual data, and at the same time aim to broaden potential applications of the techniques, incorporate them into the many different phases of software engineering processes that involve interaction with developers, managers and users, evaluate the techniques in real-world settings, and establish ties with the software industry and ultimately help the industry to speed up development, save developer time, improve software quality, and reduce bugs and maintenance costs.

Selected Publications and Outputs

[Tool/Deckard] Lingxiao Jiang. Main developer and maintainer for DECKARD: A Scalable Code Clone and Clone-Related Bug Detection Tool, open-sourced at https://github.com/skyhover/Deckard, and

EqMiner/DyClone: A dynamic code clone detection tool, open-sourced at https://github.com/skyhover/dyclone

[Tool/LibraryGuru] Weizhao Yuan, Hoang Huu Nguyen, Lingxiao Jiang, and Yuting Chen. LibraryGuru: API Recommendation for Android Developers. In the 40th International Conference on Software Engineering (ICSE) 2018 Demonstrations Track, May 27 - June 3, Gothenburg, Sweden. Tool available at: http://libraryguru.info.

[Tool/SmartEmbed] Zhipeng Gao, Vinoj Jayasundara, Lingxiao Jiang, Xin Xia, David Lo, and John C. Grundy. SmartEmbed: A Tool for Checking Smart Contracts with Structural Code Embedding. Accepted by IEEE Transactions on Software Engineering (TSE), 2020. Tool available at: https://github.com/beyondacm/SmartEmbed

[Tool/FastKLEE] Haoxin Tu, Lingxiao Jiang, Xuhua Ding, He Jiang. FastKLEE: faster symbolic execution via reducing redundant bound checking of type-safe pointers. ESEC/SIGSOFT FSE 2022: 1741-1745 https://github.com/haoxintu/FastKLEE

[Tool/MandoGuru] Hoang H. Nguyen, Nhat-Minh Nguyen, Hong-Phuc Doan, Zahra Ahmadi, Thanh-Nam Doan, and Lingxiao Jiang. MANDO-GURU series: vulnerability detection for smart contract source code and bytecode via heterogeneous graph deep graph learning, available at https://github.com/MANDO-Project Last update 2023/12.

[ASE2023] Vikas Kumar Malviya, Yan Naing Tun, Chee Wei Leow, Ailys Tee Xynyn, Lwin Khin Shar, Lingxiao Jiang: Fine-Grained In-Context Permission Classification for Android Apps Using Control-Flow Graph Embedding. ASE 2023: 1225-1237

[CCS2023] Pansilu Pitigalaarachchi, Xuhua Ding, Haiqing Qiu, Haoxin Tu, Jiaqi Hong, Lingxiao Jiang: KRover: A Symbolic Execution Engine for Dynamic Kernel Analysis. CCS 2023: 2009-2023

[ISSTA2023] Yuzhou Fang, Daoyuan Wu, Xiao Yi, Shuai Wang, Yufan Chen, Mengjie Chen, Yang Liu, Lingxiao Jiang: Beyond "Protected" and "Private": An Empirical Security Analysis of Custom Function Modifiers in Smart Contracts. ISSTA 2023: 1157-1168

[MSR2023a] Imam Nur Bani Yusuf, Diyanah Binte Abdul Jamal, Lingxiao Jiang: Automating Arduino Programming: From Hardware Setups to Sample Source Code Generation. MSR 2023: 453-464

[MSR2023b] Hoang H. Nguyen, Nhat-Minh Nguyen, Chunyao Xie, Zahra Ahmadi, Daniel Kudendo, Thanh-Nam Doan, Lingxiao Jiang: MANDO-HGT: Heterogeneous Graph Transformers for Smart Contract Vulnerability Detection. MSR 2023: 334-346

[NDSS2023] Xiao Yi, Yuzhou Fang, Daoyuan Wu, Lingxiao Jiang: BlockScope: Detecting and Investigating Propagated Vulnerabilities in Forked Blockchain Projects. NDSS 2023

[ASE2022] Vikas Kumar Malviya, Chee Wei Leow, Ashok Kasthuri, Yan Naing Tun, Lwin Khin Shar, Lingxiao Jiang: Right to Know, Right to Refuse: Towards UI Perception-Based Automated Fine-Grained Permission Controls for Android Apps. ASE 2022: 186:1-186:6

[DSAA2022] Hoang H. Nguyen, Nhat-Minh Nguyen, Chunyao Xie, Zahra Ahmadi, Daniel Kudendo, Thanh-Nam Doan, Lingxiao Jiang: MANDO: Multi-Level Heterogeneous Graph Embeddings for Fine-Grained Detection of Smart Contract Vulnerabilities. DSAA 2022: 1-10

[FSE2022] Xiao Yi, Daoyuan Wu, Lingxiao Jiang, Yuzhou Fang, Kehuan Zhang, Wei Zhang: An empirical study of blockchain system vulnerabilities: modules, types, and patterns. ESEC/SIGSOFT FSE 2022: 709-721

[AAAI2021] Nghi Duy Quoc Bui, Yijun Yu, and Lingxiao Jiang. TreeCaps: Tree-based Capsule Networks for Source Code Processing. In the 35th AAAI Conference on Artificial Intelligence (AAAI), 2021. Open-sourced available at: https://github.com/bdqnghi/treecaps

[ICSE2021] Nghi Duy Quoc Bui, Yijun Yu, and Lingxiao Jiang. InferCode: Self-Supervised Learning of Code Representations by Predicting Subtrees. In the IEEE/ACM 43th International Conference on Software Engineering (ICSE), 2021. Open-sourced available at: https://github.com/bdqnghi/infercode

[SIGIR2021] Nghi Duy Quoc Bui, Yijun Yu, and Lingxiao Jiang. Self-Supervised Contrastive Learning for Code Retrieval and Summarization via Semantic-Preserving Transformations. In 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), 2021.

[APSEC2020] Lwin Khin Shar, Ta Nguyen Binh Duong, Lingxiao Jiang, David Lo, Wei Minn, Glenn Kiah Yong Yeo, Eugene Kim. SmartFuzz: An Automated Smart Fuzzing Approach for Testing SmartThings Apps. APSEC 2020: 365-374

[ICSME2020] Muhammad Hilmi Asyrofi, Ferdian Thung, David Lo, Lingxiao Jiang. CrossASR: Efficient Differential Testing of Automatic Speech Recognition via Text-To-Speech. ICSME 2020: 640-650

[ASE2019] Nghi Duy Quoc Bui, Yijun Yu, and Lingxiao Jiang. AutoFocus: Interpreting Attention-based Neural Networks by Code Perturbation. In the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 38-41, 2019.

[FSE2019] Nghi Duy Quoc Bui, Yijun Yu, and Lingxiao Jiang. SAR: Learning Cross-Language API Mappings with Little Knowledge. In 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 796-806, 2019.

[SANER2019] Nghi Duy Quoc Bui, Yijun Yu, and Lingxiao Jiang. Bilateral Dependency Neural Networks for Cross-Language Algorithm Classification. In the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 422-433, 2019.

[ICSE2018NIER] Nghi Duy Quoc Bui and Lingxiao Jiang. Hierarchical learning of cross-language mappings through distributed vector representations for code. In the 40th International Conference on Software Engineering: New Ideas and Emerging Results, pp. 33-36, 2018.

[IEICEJournal2017] Xiao Cheng, Zhiming Peng, Lingxiao Jiang, Hao Zhong, Haibo Yu, and Jianjun Zhao. CLCMiner: Detecting Cross-Language Clones without Intermediates. In IEICE Transaction on Information and Systems, vol. 100-D(2), pp. 273-284, 2017.

[ICPC2017] Hoang Huu Nguyen, Lingxiao Jiang, Thanh Tho Quan: Android repository mining for detecting publicly accessible functions missing permission checks, In 25th IEEE International Conference on Program Comprehension (ICPC), pp. 324-327, 2017.

[AST2016] Joseph J. K. Chan, Lingxiao Jiang, Kiat Wee Tan and Rajesh K. Balan. Graph-Aided Directed Testing of Android Applications for Checking Runtime Privacy Behaviours. In 11th International Workshop on Automation of Software Test (AST), pp. 57-63, 2016.

[IWSM2016] Xiao Cheng, Lingxiao Jiang, Hao Zhong, Haibo Yu and Jianjun Zhao. On the Feasibility of Detecting Cross-Platform Code Clones via Identifier Similarity. In the proceedings of the 5th International Workshop on Software Mining (IWSM), pp. 39-42, 2016.

[ASE2016] Xiao Cheng, Zhiming Peng, Lingxiao Jiang, Hao Zhong, Haibo Yu and Jianjun Zhao. Mining Revision Histories to Detect Cross-Language Clones without Intermediates. In 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 696-701, 2016.

[HICSS2015] Youngsoo Kim and Lingxiao Jiang. The Knowledge Accumulation and Transfer in Open-Source Software (OSS) Development. In the proceedings of the 48th Hawaii International Conference on System Sciences (HICSS), pp. 3811-3820, 2015.

[ASEJournal2015] Ferdian Thung, Lucia, David Lo, Lingxiao Jiang, Foyzur Rahman and Premkumar T. Devanbu. To What Extent Could We Detect Field Defects? An Extended Empirical Study of False Negatives in Static Bug Finding Tools. Journal of Automated Software Engineering (AUSE), 22(4):561-602, 2015.

[JSEPJournal2014] Lucia, David Lo, Lingxiao Jiang, Ferdian Thung and Aditya Budi. Extended comprehensive study of association measures for fault localization. Journal of Software: Evolution and Process (JSEP), 26(2):172-219, 2014.

[ISSTA2014] Narcisa Andreea Milea, Lingxiao Jiang and Siau-Cheng Khoo. Scalable detection of missed cross-function refactorings. In International Symposium on Software Testing and Analysis (ISSTA), pp. 138-148, San Jose, CA, USA, 2014.

[FSE2014] Narcisa Andreea Milea, Lingxiao Jiang and Siau-Cheng Khoo. Vector Abstraction and Concretization for Scalable Detection of Refactorings. In 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), pp. 86-97, Hong Kong, 2014.

[ISSRE2013] Tegawende F. Bissyande, David Lo, Lingxiao Jiang, Laurent Reveillere, Jacques Klein and Yves Le Traon. Got Issues? Who Cares About It? A Large Scale Investigation of Issue Trackers from GitHub. In the proceedings of the IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), pp. 188-197, 2013.

[QSIC2013] Pavneet Singh Kochhar, Tegawende F. Bissyande, David Lo and Lingxiao Jiang. An Empirical Study of Adoption of Software Testing in Open Source Projects. In the proceedings of the 13th International Conference on Quality Software (QSIC), pp. 103-112, 2013.

[CSMR2013a] Ferdian Thung, Tegawende F. Bissyande, David Lo and Lingxiao Jiang. Network Structure of Social Coding in GitHub. In the proceedings of the 17th European Conference on Software Maintenance and Reengineering (CSMR), pp. 323-326, Genova, Italy, 2013.

[SAC2013] Shaowei Wang, David Lo and Lingxiao Jiang. An Empirical Study on Developer Interactions in StackOverflow. In the 28th Symposium on Applied Computing (SAC), pp. 1019-1024, Coimbra, Portugal, 2013.

[CSMR2013b] Shaowei Wang, David Lo and Lingxiao Jiang. Understanding Widespread Changes: A Taxonomic Study. In the proceedings of the 17th European Conference on Software Maintenance and Reengineering (CSMR), pp. 5-14, Genova, Italy, 2013.

[ICSE2012] Lucia, David Lo, Lingxiao Jiang and Aditya Budi. Active refinement of clone anomaly reports. In 34th International Conference on Software Engineering (ICSE), pp. 397-407, 2012.

[ASE2012a] Lucia, David Lo, Lingxiao Jiang and Aditya Budi. kbe-Anonymity: Test Data Anonymization for Evolving Programs. In the proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 262-265, 2012.

[ICSM2012] Ferdian Thung, David Lo and Lingxiao Jiang. Detecting Similar Applications with Collaborative Tagging. In the proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM), pp. 600-603, 2012.

[APSEC2012] Ferdian Thung, David Lo and Lingxiao Jiang. Diffusion of Software Features: An Exploratory Study. In the 19th Asia-Pacific Software Engineering Conference (APSEC), pp. 368-373, 2012.

[ASE2012b] Ferdian Thung, Lucia, David Lo, Lingxiao Jiang, Foyzur Rahman and Premkumar T. Devanbu. To What Extent Could We Detect Field Defects? ---An Empirical Study of False Negatives in Static Bug Finding Tools. In the proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 50-59, 2012.

[PLDI2011] 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 conference on Programming Language Design and Implementation (PLDI), pp. 447-457, 2011.

[WCRE2011a] Shaowei Wang, David Lo and Lingxiao Jiang. Code Search via Topic-Enriched Dependence Graph Matching. In the proceedings of the 18th Working Conference on Reverse Engineering (WCRE), pp. 119-123, 2011.

[ASE2011] 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), pp. 556-559, Lawrence, Kansas, USA, 2011.

[WCRE2011b] 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 Working Conference on Reverse Engineering (WCRE), pp. 92-96, 2011.

[ICSM2010] Lucia, David Lo, Lingxiao Jiang and Aditya Budi. Comprehensive Evaluation of Association Measures for Fault Localization. In the proceedings of the IEEE International Conference on Software Maintenance (ICSM), pp. 1-10, Timisoara, Romania, 2010.

[ISSTA2009] Lingxiao Jiang and Zhendong Su. Automatic mining of functionally equivalent code fragments via random testing. In 18th International Symposium on Software Testing and Analysis (ISST)A, pp. 81-92, 2009.

[ICSE2008] Mark Gabel, Lingxiao Jiang and Zhendong Su. Scalable detection of semantic clones. In the International Conference on Software Engineering (ICSE), pp. 321-330, 2008.

[FSE2008] 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), pp. 48-58, 2008.

[ICSE2007] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su and Stephane Glondu. DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones. In 29th International Conference on Software Engineering (ICSE), pp. 96-105, 2007.

[ASE2007] Lingxiao Jiang and Zhendong Su. Context-aware statistical debugging: from bug predictors to faulty control flow paths. In 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 184-193, 2007.

[FSE2007] Lingxiao Jiang, Zhendong Su and Edwin Chiu. Context-based detection of clone-related bugs. In the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 55-64, 2007.