Research Statement

I lead the **Software Evolution and Analysis (SEA) Lab** where I currently mentor five Ph.D. students and one M.Sc. student. My **research interests** are in software engineering (SE), focusing on supporting the evolution and maintenance of large-scale software systems. I work on developing techniques, tools, and methodologies that help software developers understand, build, and maintain high-quality software.

My research analyzes and leverages content from different *software artifacts* (bug reports, source code, online discussions, *etc.*) and builds on creating, adapting, and integrating *techniques* based on program analysis, software repository mining (MSR), information retrieval (IR), natural language processing (NLP), and machine/deep learning (ML/DL). These techniques are typically informed by and evaluated via *empirical studies* such as simulations, case studies, and user studies. Our techniques are often implemented and released as open source software tools. Associated benchmarks, including datasets, infrastructure, and documentation, are often released to the SE community to foster further advances in SE.

My **current research areas** are on: **(i)** automated bug report management [1–3, 6, 8–14, 16–21, 23, 25–28, 32], **(ii)** verification-guided code refactoring and comprehension [5, 15], **(iii)** software licensing and supply chain management [29–31], and **(iv)** informed decision making for software change [15, 23]. Additional areas I have worked on include software documentation [22, 34], quantum software development [33], and software reverse engineering [4, 7].

My research has been published at **premier SE conferences and journals**, including the IEEE/ACM International Conference on Software Engineering (ICSE) [2, 3, 13, 14, 17, 21, 27, 29, 31, 32], the ACM International Conference on the Foundations of Software Engineering (ESEC/FSE or FSE) [6, 12, 15, 25, 28], the IEEE International Conference on Software Maintenance and Evolution (ICSME) [5, 8, 9], the IEEE Transactions on Software Engineering (TSE) journal [1], and the Empirical Software Engineering (EMSE) journal [10]. According to Google Scholar, **my publications have been cited 933 times** as of May 9th, 2024 (826 times since 2019), with an H-index of 16 (14 since 2019), and an I10 index of 19 (18 since 2019).

I have earned **multiple awards** that recognize the quality of my research and publications, including an NSF CAREER Award, two ACM SIGSOFT Distinguished Paper Awards at ICSE'20 [2] and FSE'19 [6], one IEEE TCSE Distinguished Paper Award at ICSME'17 [9], and the 2019 David Daniel Thesis Award from my *alma mater*, The University of Texas at Dallas.

1 Automated Bug Report Management

Software systems often contain bugs (*a.k.a.* defects) that produce unexpected results and negatively affect user productivity and experience. Most of these bugs are reported manually via issue trackers, as they do not lead to an explicit system failure. Bug reports describe the defects found during software development and usage, and are the main artifacts for developers to triage, replicate, and correct the reported bugs. Despite their importance, bug reports are often unclear, incomplete, and hard to understand, as reported by numerous studies (including ours [9, 10, 12, 25]). Low-quality bug reports result in excessive manual effort spent on bug triage and resolution, and even in the inability of developers to identify, reproduce, and correct the reported bugs.

The excessive manual effort of issue resolution is intensified by the large number of reports received daily, especially in large software projects, which developers have to manually understand, analyze, and manage. As such, bug report management is often a time-consuming and error-prone process; developers need automated support in tasks such as bug reporting, triage, localization, replication, and fixing. While researchers have worked on automating these tasks, there is still a long path in this area as the accuracy, effectiveness, and usability of many proposed techniques are rather limited.

The main goal of our research is to improve how users and developers report, triage, replicate, and solve software bugs. To this end, we have studied the process of bug report management and have developed techniques and tools that are more effective, accurate, and useful than the state of the art.

The Discourse of Bug Descriptions. We empirically verified the hypothesis that reporters use a narrow discourse to describe software problems in textual bug reports, which enables the automated analysis of such reports to support bug management tasks. We conducted a study to analyze the syntax and semantics of sentences and paragraphs that describe the main information elements for developers: the observed (unexpected) software behavior (OB), the steps to reproduce such behavior (S2Rs), and the expected software behavior (EB). Based on qualitative analysis of a large set of bug reports from nine large software systems, we found that reporters use only 154 patterns to describe the OB, EB, and S2Rs. These results were published at FSE'17 [12].

Bug Report Quality Assessment. Bug reports often lack essential information for developers [9, 10, 12, 25]. Indeed, our FSE'17 work [12] found that only 35% and 51% of submitted bug reports describe the EB and S2Rs. Based on the regularities found in bug descriptions, which enable the accurate prediction of the absence (or presence) of the OB, EB, and S2Rs, we developed an automatic detector based on machine learning that utilizes the identified discourse patterns and additional textual features to detect when EB and S2Rs are missing in bug reports [12]. We found that our technique can detect missing EB and S2Rs with high accuracy: 69% and 86% precision and 83% and 94% recall, respectively, outperforming baseline approaches. Our technique can be deployed on new software projects without needing to retrain our model, and still achieve high detection accuracy. Based on these results, we developed a machine-learning-based GitHub plugin, called BEE (published at FSE'20 [25]), that identifies individual OB, EB, and S2R sentences in textual bug reports, alerts reporters about missing information, and provides a web API that enables further research in bug report management.

In follow-up work, we proposed EULER, an automated approach that analyzes a textual bug report, identifies the provided S2Rs, assesses the quality of each step, and generates actionable feedback to reporters about (1) ambiguous steps, (2) steps described with unexpected vocabulary, and (3) missing individual steps. EULER combines NLP, ML, dynamic software analysis, and textual matching techniques to automatically identify the S2Rs and assign to each step quality annotations with specific feedback to the reporter. EULER's feedback was assessed by external evaluators in an empirical evaluation. We found that EULER correctly identified 98% of the existing S2Rs and inferred 58% of the missing ones (with a 31% precision), while 73% of its quality annotations are correct. We found evidence that EULER can be useful in helping reporters improve their bug reports. This research was published at FSE'19 [6], winning an **ACM SIGSOFT Distinguished Paper Award.**

Interactive Bug Reporting. As a paradigm shift from static to interactive bug reporting, we proposed BURT, a web-based chatbot for interactive reporting of Android app bugs. BURT combines techniques based on NLP, ML, dynamic software analysis, and textual matching, to (i) guide the user in reporting essential bug report elements (OB, EB, and S2Rs), (ii) check the quality of these elements, (iii) offer instant feedback about issues in the descriptions, and (iv) provide graphical suggestions (*e.g.*, the next S2Rs). BURT's evaluation with end-users, who reported several bugs from different Android app bugs, found that BURT's guidance and automated suggestions/clarifications are useful and easy to use, and the resulting bug reports are of higher quality than those reported with traditional issue trackers. This work was published at FSE'22 [28] and ICSE'23 [27].

Automated Analysis of Video-based Bug Reports. Crowdsourced testing and bug reporting technology for mobile apps allow end-users, developers, and testers to report app problems via video screen recordings, which depict usage scenarios and unexpected app behavior (*i.e.*, video-based bug reports). Given the increasing integration of screen capture technology into mobile apps and issue trackers, and the fact that mobile app projects often receive numerous incoming videos, developers face challenges in replicating the reported issues and determining if a given bug was reported before.

To support developers in bug replication and app testing, we proposed V2S, an automated approach that replicates the app usage scenario shown in a video recording. Based on computer vision techniques and automated object identification in video frames, V2S detects the actions that users perform on a mobile app and converts these actions into low-level commands. When executed, the commands replicate the scenario shown

in the recording. The evaluation we conducted with popular Android apps revealed that V2S achieves high detection and replication of the app usage scenarios depicted in the videos. This work won an **ACM SIGSOFT Distinguished Paper Award** at ICSE'20 [2], and was extended for publication in the TSE journal [1].

To assist developers in detecting duplicate video-based bug reports, we proposed Tango and Janus, two approaches based on tailored computer vision models (*e.g.*, Vision Transformer) and information retrieval techniques that leverage textual, visual, and sequential information found in video frames. We evaluated the approaches on collected video-based data for a set of Android apps, and found it is highly accurate in this task. This work was published at ICSE'21 [14] and ICSE'24 [32].

Enhanced Bug Localization and Duplicate Detection. Many existing techniques use the full text in bug reports as input queries (*a.k.a.* initial queries) to text retrieval (TR) techniques for (1) detecting if they are duplicates of existing reports (duplicate detection) and (2) finding the source code that likely contains the reported bug (bug localization). These techniques are meant to avoid potentially unnecessary work (*i.e.*, triage/solve the same bugs over and over) and help developers locate the buggy code. These techniques typically return a ranked list of potential duplicate reports or buggy code artifacts (*e.g.*, classes), where a higher rank for such elements indicates a higher relevance.

Unfortunately, the initial queries often contain noisy information that leads to low vocabulary agreement [8], failing to retrieve the duplicate reports and the buggy code artifacts. Our solution to this problem is query reduction [3], which consists of removing bug report terms that are noisier than others, and using the remaining, more relevant text as input queries. We verified the hypothesis that the OB, EB, and S2Rs content is more relevant than other terms, and proposed a set of strategies for reformulating the initial queries based on the combination of OB, EB, S2Rs, and other information. We found that these strategies significantly improve the detection performance of existing techniques across multiple data sets for both bug localization and duplicate detection. The results of this work were published at several SE venues [3, 8, 10, 11, 13, 16], including ICSME'17 [9], where we received an **IEEE TCSE Distinguished Paper Award**. Our follow-up research, published at SANER'21 [16], found higher bug localization performance when combining query reduction and expansion, by using task descriptions in bug reports as part of the reformulated queries.

In our recent ICSE'24 work [21], we found that leveraging GUI interaction information from mobile app screens (*e.g.*, metadata from interacted GUI screens and widgets) as input to TR and neural bug localizers leads to significant bug localization improvements. Our ISSTA'24 work [24] found that existing textual and multi-modal neural models are useful to automatically identify buggy GUI screens and widgets to fully automate our ICSE'24 buggy code localization approach.

Future Research Agenda. My long-term research will focus on radically transforming the way software bugs are reported, triaged, and solved, via accurate and interactive recommender tools that cater to specific stakeholders and contexts. We aim to design interactive bug reporting, triage, and resolution tools for different types of users (*e.g.*, novice and experienced developers), software systems (*e.g.*, web systems and libraries), and bugs (*e.g.*, incorrect output and navigation issues), considering user context and preferences.

2 Verification-guided Code Refactoring and Comprehension

Software developers must deeply understand source code to implement new features, fix defects, review and refactor code, and perform other SE tasks. Unfortunately, understanding code is challenging and time-consuming. Prior studies estimate that developers spend 58% to 70% of their time only in understanding code. Complexity is a major reason why source code can be hard to understand: code might be written in convoluted ways or be composed of interacting code structures and dependencies that are difficult to understand. Researchers have proposed many syntactic metrics that aim to measure code complexity, but the correlation between such metrics and the measured comprehensibility of code derived from humans is weak at best. One reason for this low correlation is the fact that such metrics may not capture code semantic properties.

Since formal code verification techniques (*i.e.*, verifiers) reason about code semantics to verify code correctness, we hypothesized that verifiers produce more false positives on code that is complex than on code that is simple: complex code is harder to model. We conducted a statistical meta-analysis (published at FSE'23 [15]) to validate this hypothesis and found a positive correlation between the effort to verify code by using verification tools (*a.k.a.* code verifiability), as measured by the number of false positive warnings, and the effort to understand code by humans, as measured by different proxies used in prior studies with human subjects (*e.g.*, determining program output or measuring the time to understand code). These results imply that: (1) code that is easier to verify is often easier to understand by humans, (2) code semantics might be a key factor for code complexity, (3) code semantics information produced by verification tools could be leveraged for measuring code comprehensibility more effectively, and (4) reducing the number of verifier warnings via code refactoring could help reduce code complexity and understandability.

Future Research Agenda. Based on these results, we propose to (1) conduct studies to further validate our hypothesis and establish whether a causal relationship exists between complexity, verifiability, and comprehensibility; (2) use the semantic code information captured by verifiers to improve the performance of existing predictive models of comprehensibility that currently rely on syntactic features only; and (3) develop verification-guided refactoring techniques to reduce complexity and comprehensibility automatically, using information about the parts of programs that verifiers struggle with as a guide for where and how to apply refactoring.

3 Software Licensing and Supply Chain Management

Software systems are often composed of numerous third-party components, often from the open-source software (OSS) community. Such component reuse has created a dynamic supply chain in which developers freely utilize existing solutions to common problems and thus accomplish their tasks more productively. However, leveraging external packages does not come without a cost. The fate of a software product is intrinsically tied to its evolving dependencies. If a dependency displays a security vulnerability, then so too could the final product. Moreover, failing to comply with the license terms of software dependencies could result in significant legal, reputational, and financial consequences for developers and organizations. These problems are amplified in software projects that include code from programming forums (*e.g.*, StackOverflow) or code produced by generative AI systems (*e.g.*, ChatGPT) [30]. As software is subject to copyright law, (re)using code without permission to train generative AI models or develop derivative systems may lead to major legal risks.

In this context, it is essential to understand and address the challenges that developers and other stakeholders (*e.g.*, legal practitioners) face in supply chain management activities, including reusing software dependencies, tracking dependencies and their origins, interpreting licensing terms, preventing and correcting licensing issues, and mitigating legal and security risks.

We conducted a large-scale user study to understand the challenges developers face when creating and using Software Bills of Materials (SBOMs). SBOMs are machine-readable manifests, meant to contain comprehensive software dependency information, leading to improved vulnerability management, license compliance, and transparency in the supply chain. We surveyed 138 practitioners, including developers familiar with SBOMs, members of critical OSS projects, AI/ML developers, experts in cyber-physical systems, and a legal practitioner. We identified 12 major challenges, including inaccurate and incomplete SBOM creation due to deficiencies in current SBOM tooling, difficult SBOM maintenance and verification for evolving dependencies, lack of support for ecosystems with no package managers, inability to determine origins of dependencies and code, and unclear SBOM direction and low adoption. We proposed actionable solutions that address these challenges, including developing multi-dimensional SBOM specifications applicable to different software kinds, SBOM tools integrated with build systems and compilers, strategies for SBOM verification, and increased incentives for SBOM adoption. This work was published at ICSE'24 [29].

Legal practitioners play a critical role in the process of license compliance within organizations. However,

little is known in the SE community about their experience in performing license compliance. We conducted a qualitative study that examined the experiences of 30 legal practitioners specializing in OSS license compliance in the U.S. The study yielded 14 main findings about OSS license compliance, which highlight the need for: (1) robust tooling that can perform compliance checks at scale, for software with hundreds or thousands of components, while minimizing false positives and negatives; (2) effective communication and interaction between lawyers and developers as legal compliance is an integral part of software engineering; (3) continuous license compliance, integrated into the software development process; and (4) better support to understand licenses, especially the interpretation of gray areas. This work was accepted for publication at FSE'24 [31].

Future Research Agenda. We will focus on (1) automatically identifying dependencies and tracking their origins, to generate accurate and verifiable SBOMs that can be attached to binaries and can easily evolve whenever dependencies change in a project; and (2) developing interactive techniques that comprehensibly analyze SBOMs to provide answers to stakeholders about license compliance and vulnerability risks.

4 Informed Decision Making for Software Change

Software developers continuously change source code to add new functionality and make improvements to software systems. Unfortunately, during this process, developers often make code changes that increase software defects, unintended system behavior, and code deterioration. A crucial cause for low-quality code changes is the fact that developers often have insufficient knowledge about the code, which leads them to make poor decisions on how to correctly change it. Obtaining such knowledge is extremely challenging, in part because documented code-related information is unstructured, fragmented, and scattered across various software artifacts without explicit traceability links among them. To address these fundamental challenges, our research program will process and manage code change decisions documented in software artifacts/repositories to assist developers in making informed decisions on how to correctly change source code. The premise that guides this research, supported by prior results, is that documented code change decisions contain valuable code-related knowledge that can inform developers in designing and implementing code changes that meet requirements and minimize the introduction of defects, unintended system behavior, and code decay.

Future Research Agenda. With this in mind, we aim to develop a theory of code change decisions that will document: (i) strategies and patterns of decision-making, (ii) factors that make adequate and poor code change decisions, and (iii) actionable guidelines on how developers should make/reuse code change decisions to solve new problems. We will design and develop novel automated techniques and interactive tool support for capturing and tracing information elements of code change decisions, while developers document code-related knowledge in various software artifacts. Finally, we will design and develop automated techniques and interactive tool support to inform developers about: (i) the reasons why past code changes were made, (ii) past decisions relevant to solve a change request or defect report, and (iii) evidence of the impact that past decisions had on code quality and defect introduction. The proposed theory and techniques will be developed through cross-cutting research on empirical software engineering, automated text analysis, machine/deep learning, information retrieval, and human-computer interaction. The results of our research will allow developers to easily capture and manage their decisions in software artifacts while solving new change requests and defect reports. Developers will better learn from prior decisions to produce software that is less faulty and easier to maintain.

Initial Results. A key step in this research was a case study that investigated on how Mozilla Firefox developers address issues submitted in issue trackers, including how they analyze the issues, design solutions, and make decisions on how to change the code. We qualitatively annotated and analyzed the comments written by developers in 356 issue reports, and derived a catalog of 47 issue resolution patterns recurrently used to address various software problems. The patterns vary in complexity and are employed to address different problems (*e.g.*, UI issues, refactoring, and incorrect functionality) and issue types (*e.g.*, defects, enhancements, and tasks). This work is under review at ICSME'24 [23].

References

[1] C. Bernal-Cárdenas, N. Cooper, M. Havranek, K. Moran, O. Chaparro, D. Poshyvanyk, and A. Marcus. Translating video recordings of complex mobile app UI gestures into replayable scenarios. *IEEE Transactions on Software Engineering (**TSE**)*, 49(4):1782–1803, 2023.

[2] C. Bernal-Cárdenas, N. Cooper, K. Moran, O. Chaparro, A. Marcus, and D. Poshyvanyk. Translating video recordings of mobile app usages into replayable scenarios. In *Proceedings of the 42nd IEEE/ACM International Conference on Software Engineering (**ICSE'20**)*, page 309–321, 2020. **ACM SIGSOFT Distinguished Paper Award**.

[3] O. Chaparro. Improving bug reporting, duplicate detection, and localization. In *Proceedings of the 39th ACM/IEEE International Conference on Software Engineering (**ICSE'17**)*, page 421–424, 2017.

[4] O. Chaparro, J. Aponte, F. Ortega, and A. Marcus. Towards the automatic extraction of structural business rules from legacy databases. In *Proceedings of the 19th IEEE Working Conference on Reverse Engineering (**WCRE'12**)*, page 479–488, 2012.

[5] O. Chaparro, G. Bavota, A. Marcus, and M. Di Penta. On the impact of refactoring operations on code quality metrics. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution (**ICSME'14**)*, page 456–460, 2014.

[6] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, and V. Ng. Assessing the quality of the steps to reproduce in bug reports. In *Proceedings of the 27th ACM Joint Meeting on the Foundations of Software Engineering (**ESEC/FSE'19**)*, page 86–96, 2019. **ACM SIGSOFT Distinguished Paper Award**.

[7] O. Chaparro, F. Cortés, and J. Aponte. Reverse engineering in procedural software evolution. In *Research topics in software evolution and maintenance*, pages 127–153. Editorial Universidad Nacional de Colombia, 2012.

[8] O. Chaparro, J. M. Florez, and A. Marcus. On the vocabulary agreement in software issue descriptions. In *Proceedings of the 32nd IEEE International Conference on Software Maintenance and Evolution (**ICSME'16**)*, page 448–452, 2016.

[9] O. Chaparro, J. M. Florez, and A. Marcus. Using observed behavior to reformulate queries during text retrieval-based bug localization. In *Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution (**ICSME'17**)*, page 376–387, 2017. **IEEE TCSE Distinguished Paper Award**.

[10] O. Chaparro, J. M. Florez, and A. Marcus. Using bug descriptions to reformulate queries during text-retrieval-based bug localization. *Empirical Software Engineering (**EMSE**)*, 24(5):2947–3007, 2019.

[11] O. Chaparro, J. M. Florez, U. Singh, and A. Marcus. Reformulating queries for duplicate bug report detection. In *Proceedings of the IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (**SANER'19**)*, page 218–229, 2019.

[12] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng. Detecting missing information in bug descriptions. In *Proceedings of the 11th ACM Joint Meeting on the Foundations of Software Engineering (**ESEC/FSE'17**)*, page 396–407, 2017.

[13] O. Chaparro and A. Marcus. On the reduction of verbose queries in text retrieval based software maintenance. In *Proceedings of the 38th ACM/IEEE International Conference on Software Engineering (**ICSE'16**)*, page 716–718, 2016.

[14] N. Cooper, C. Bernal-Cárdenas, O. Chaparro, K. Moran, and D. Poshyvanyk. It takes two to tango: Combining visual and textual information for detecting duplicate video-based bug reports. In *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (**ICSE'21**)*, page 957–969, 2021.

[15] K. Feldman, M. Kellogg, and O. Chaparro. On the relationship between code verifiability and understandability. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (**ESEC/FSE'23**)*, page 211–223, 2023.

[16] J. M. Florez, O. Chaparro, C. Treude, and A. Marcus. Combining query reduction and expansion for text-retrieval-based bug localization. In *Proceedings of the 28th IEEE International Conference on Software Analysis, Evolution and Reengineering (**SANER'21**)*, page 166–176, 2021.

[17] M. Havranek, C. Bernal-Cárdenas, N. Cooper, O. Chaparro, D. Poshyvanyk, and K. Moran. V2S: A tool for translating video recordings of mobile app usages into replayable scenarios. In *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (**ICSE'21**)*, page 65–68, 2021.

[18] R. Kallis, O. Chaparro, A. Di Sorbo, and S. Panichella. NLBSE'22 tool competition. In *Proceedings of the 1st IEEE/ACM International Workshop on Natural Language-Based Software Engineering (**NLBSE'22**)*, page 25–28, 2022.

[19] R. Kallis, G. Colavito, A. Al-Kaswan, L. Pascarella, O. Chaparro, and P. Rani. The NLBSE'24 tool competition. In *Proceedings of the 3rd IEEE/ACM International Workshop on Natural Language-Based Software Engineering (**NLBSE'24**)*, page 25–28, 2024.

[20] R. Kallis, M. Izadi, L. Pascarella, O. Chaparro, and P. Rani. The NLBSE'23 tool competition. In *Proceedings of the 2nd IEEE/ACM International Workshop on Natural Language-Based Software Engineering (**NLBSE'23**)*, page 25–28, 2023.

[21] J. Mahmud, N. De Silva, S. A. Khan, S. H. Mostafavi, S. M. H. Mansur, O. Chaparro, A. A. Marcus, and K. Moran. On using GUI interaction data to improve text retrieval-based bug localization. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (**ICSE'24**)*, page 1–13, 2024.

[22] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong. On-demand developer documentation. In *Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution (**ICSME'17**)*, page 479–483, 2017.

[23] A. Saha and O. Chaparro. Decoding the issue resolution process documented in issue reports: A case study of mozilla firefox. In *the 40th IEEE International Conference on Software Maintenance and Evolution (**ICSME'24**)*, (under review), 2024.

[24] A. Saha, Y. Song, J. Mahmud, Y. Zhou, K. Moran, and O. Chaparro. Toward the automated localization of buggy mobile app UIs from bug descriptions. In *the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (**ISSTA'24**)*, (major revision under review), 2024.

[25] Y. Song and O. Chaparro. BEE: a tool for structuring and analyzing bug reports. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (**ESEC/FSE'20**)*, page 1551–1555, 2020.

[26] Y. Song and O. Chaparro. Recommending bug assignment approaches for individual bug reports: An empirical investigation. (arXiv:2305.18650), 2023. arXiv:2305.18650 [cs].

[27] Y. Song, J. Mahmud, N. De Silva, Y. Zhou, O. Chaparro, K. Moran, A. Marcus, and D. Poshyvanyk. BURT: a chatbot for interactive bug reporting. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (**ICSE'23**)*, page 170–174, 2023.

[28] Y. Song, J. Mahmud, Y. Zhou, O. Chaparro, K. Moran, A. Marcus, and D. Poshyvanyk. Toward interactive bug reporting for (Android app) end-users. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (**ESEC/FSE'22**)*, page 344–356, 2022.

[29] T. Stalnaker, N. Wintersgill, O. Chaparro, M. Di Penta, D. M. German, and D. Poshyvanyk. BOMs away! inside the minds of stakeholders: A comprehensive study of bills of materials for software systems. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (**ICSE'24**)*, page 1–13, 2024.

[30] T. Stalnaker, N. Wintersgill, O. Chaparro, L. A. Heymann, M. Di Penta, D. M. German, and D. Poshyvanyk. Licensing and copyright issues in using generative AI for coding: A practitioner perspective. In *the IEEE/ACM 47th International Conference on Software Engineering (**ICSE'25**)*, (under review), 2025.

[31] N. Wintersgill, T. Stalnaker, L. A. Heymann, O. Chaparro, and D. Poshyvanyk. "the law doesn't work like a computer": Exploring software licensing issues faced by legal practitioners. In *Proceedings of the ACM International Conference on the Foundations of Software Engineering (**FSE'24**)*, page (to appear), 2024.

[32] Y. Yan, N. Cooper, O. Chaparro, K. Moran, and D. Poshyvanyk. Semantic GUI scene learning and video alignment for detecting duplicate video-based bug reports. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (**ICSE'24**)*, page 1–13, 2024.

[33] J. Zappin, T. Stalnaker, O. Chaparro, and D. Poshyvanyk. When quantum meets classical: Characterizing hybrid quantum-classical issues discussed in developer forums. In *the IEEE/ACM 47th International Conference on Software Engineering (**ICSE'25**)*, (under review), 2025.

[34] Y. Zhou, M. Miao, V. Birsan, O. Chaparro, S. Wei, and A. Marcus. Towards the automated identification of data constraints in software documents. In *the IEEE/ACM 47th International Conference on Software Engineering (**ICSE'25**)*, (under review), 2025.