Understanding and Improving Code Review of Changes in Build Systems

Mahtab Nejati
Software REBELs
University of Waterloo, Canada
mahtab.nejati@uwaterloo.ca

***Abstract*—Build systems orchestrate the transformation of software sources into deliverable artifacts. As a component of the ever-changing software system, they must be maintained alongside the software they build to ensure their consistency with the sources. Lax maintenance of build systems can lead to their quality decay, causing costly consequences. However, maintaining build systems is known to be challenging.**

Ensuring the quality of build systems throughout their lifespan demands rigorous quality assurance practices. As automated quality assurance methods, such as testing, are rarely applied to build systems, code review becomes a critical mechanism for safeguarding the reliability and correctness of build systems.

This thesis investigates the practices used in reviewing changes to build systems and the challenges that impede their effective code reviews (i). It introduces Build Change Impact Analysis (BCIA) as a method to facilitate code review of build systems and examines the applicability of this approach (ii). Lastly, it evaluates the effectiveness of using BCIA to improve the review process of build system changes (iii).

I. Introduction

Build tools manage the process through which source code is transformed into software deliverables, such as executables. The build process is configured in build systems, a set of build specifications that describe the internal and external dependencies among the software sources and set up the build dynamics. Build tools process build specifications to determine the order- and configuration-dependent commands that must be invoked to produce the deliverables. Build systems are supported by a variety of build technologies such as CMake.[1]

Maintenance of build systems introduces a substantial overhead on software development. Prior work [1] shows that build maintenance imposes an overhead in up to 27% and 44% of change sets in source and test code respectively. This significantly impacts 79% of source code developers and 89% of test code developers [1]. Moreover, maintaining build systems supported by modern technologies incurs an increased maintenance effort [2].

The continual changes in build systems make them susceptible to quality decay [3], which can slow down builds [4], cause build failures [5], or even lead to erroneous software behavior [6]. Despite these risks, systematic and automated quality assurance practices, such as automated testing, are rarely (if ever) applied to build specifications.[2] This leaves manual quality assurance practices as the primary approach for sustaining the quality of build systems.

Code review, i.e., the practice of developers critiquing each others' change sets, is the quintessential manual quality assurance method in modern software development. Its technical and non-technical benefits are well-documented in the literature [10], [11], including increases in code quality, peer mentorship, and knowledge transfer.

Prior work shows that code review is not applied with equal rigour across different types of software artifacts [12]. Inspired by this evidence, we seek to test the following hypothesis:

Research Hypothesis

Code review, is not extensively implemented for changes in build specifications due to challenges specific to the maintenance of build systems. Build Change Impact Analysis (BCIA) is applicable to this problem. It assists reviewers in their build code review tasks by easing their perceived cognitive load.

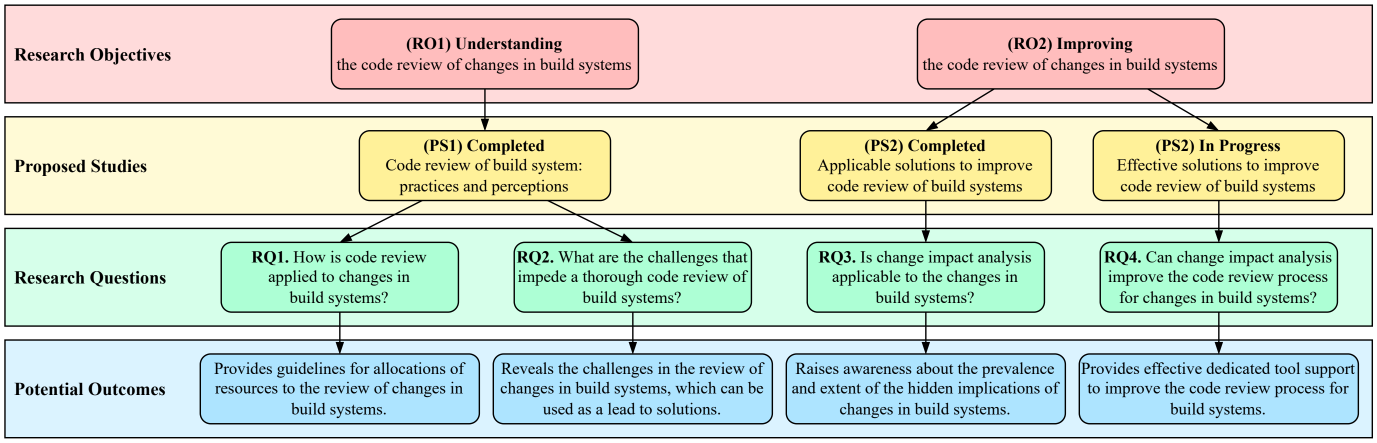
To do so, as shown in Figure 1, in this thesis, we pursue the following research objectives:

(RO1) Understand the code review of changes in build systems. Code review, as a well-established manual quality assurance method, plays a key role in sustaining the quality of build systems. However, it is still unclear whether this practice is applied to build specifications. We suspect that similar to test code [12], build system changes are less likely to receive thorough discussions during the code review process. To test this hypothesis, we perform both quantitative and qualitative analyses of real-world code review discussions [13]. Our results show that changes to build systems are at least two times less frequently discussed during code review when compared to production and test code. Follow-up interviews hint at social and technical challenges specific to the review of changes in build systems.

(RO2) Improve the code review of changes in build systems. In our work [13], practitioners complain about a perceived lack of dedicated tools to support the maintenance of build

[1] https://cmake.org/

[2] Many studies propose testing approaches for build specifications [7]–[9], However, to the best of our knowledge, they have yet to make their way to production-ready tools.



Research Objectives
(RO1) Understanding the code review of changes in build systems
(RO2) Improving the code review of changes in build systems
Proposed Studies
(PS1) Completed Code review of build system: practices and perceptions
(PS2) Completed Applicable solutions to improve code review of build systems
(PS2) In Progress Effective solutions to improve code review of build systems
Research Questions
RQ1. How is code review applied to changes in build systems?
RQ2. What are the challenges that impede a thorough code review of build systems?
RQ3. Is change impact analysis applicable to the changes in build systems?
RQ4. Can change impact analysis improve the code review process for changes in build systems?
Potential Outcomes
Provides guidelines for allocations of resources to the review of changes in build systems.
Reveals the challenges in the review of changes in build systems, which can be used as a lead to solutions.
Raises awareness about the prevalence and extent of the hidden implications of changes in build systems.
Provides effective dedicated tool support to improve the code review process for build systems.

Fig. 1: An overview of our research thesis.

systems and the review of their changes. The limited support tools leaves it a challenging task for reviewers to manually inspect and understand the implications of changes in the complex build configuration space [14]. *Change Impact Analysis (CIA)* has shown potential in improving the effectiveness and efficiency of code reviews in the context of production code by exposing the impact of changes across the system [15], [16]. Inspired by the benefits of CIA, we propose *Build Change Impact Analysis (BCIA)* [17]—an approach that uses data and control flow analysis to assess the impact of changes on the build configuration space. We implement *BuiScout*,[3] a prototype tool that applies BCIA to CMake-based build systems, and evaluate the following:

Applicability of the solution. We evaluate the applicability of BCIA in an empirical study of real-world change sets that modify build systems. Our study [17] suggests that in a majority of these change sets, an impact propagates to unmodified areas of the build system. We also find that the impact from a change is often of non-negligible magnitude and resides in parts of the build system that are non-local to the modifications. These results highlight the potential of BCIA in alleviating the challenges reviewers face when assessing the impact of changes to build systems.

Effectiveness of the solution. Applying CIA to changes in production code has shown promising improvements in assisting developers in finding the unintended and undesirable impact of changes during code review [15]. As build specifications are inherently different from production code due to their declarative nature, we investigate the practicality of BCIA in their code review. To do so, we are currently conducting user studies, using BuiScout, to evaluate the effectiveness of BCIA in real-world code review scenarios.

II. Related Work

This section situates this thesis within the existing body of literature. We organize the section along three main themes: studies on the maintenance of build systems, code review, and change impact analysis.

[3]https://github.com/mahtab-nejati/BuiScout/releases/tag/ASE2024

A. Maintenance of Build Systems

Studies have shown that extensive effort goes into the maintenance of build systems [1]. Recent work has focused on supporting the maintenance of build systems through automated defect detection [7], [9], [18] and repair [19]–[21]. While the prior work proposes promising directions, they have not yet been implemented into practice-ready tools due to their limited effectiveness [21] and performance [8], [22]. As a result, quality assurance of build systems is still predominantly a manual process, imposing a substantial cognitive load on build maintainers [14].

Another recent line of work assists build maintenance in other ways. For example, approaches have been proposed to visualize the dependency graph to clarify interdependencies between components [23], automate common refactoring tasks, such as renaming and removing targets [18], [24], detect the addition and removal of dependencies [24], and summarize reports of build failures for debugging purposes [23], [25].

Our work complements prior studies with a focus on code review of changes in build systems. We investigate the extent to which code review is leveraged to improve the maintenance of build specifications and aim to propose applicable and effective solutions that improve this process.

B. Code Review

Code review boasts numerous technical and non-technical benefits such as improved code stability, early detection of bugs, increased software quality, and improved team communication [10], [11], [26], [27]. Despite its plethora of benefits, Spadini et al. [12] found that code review is not applied with equal rigour to test code when compared to production code. Moreover, code review introduces a considerable overhead on reviewers' workload [11]. Static analysis tools can remedy the situation by providing further insights for a better understanding of the change [28].

Inspired by Spadini et al.'s work, we propose to study the code review process in the context of build specifications to better understand the extent, purposes, and challenges of its application on build system changes. We then set out to improve the code review process for build system changes by proposing BCIA, a static build change impact analysis method. We then evaluate the applicability and effectiveness of BCIA in real-world code review settings.

C. Change Impact Analysis

Change Impact Analysis (CIA) is the process of identifying and assessing the consequences of a change to the software to reveal their broader implications [29]. Prior studies have effectively implemented CIA techniques into processes that require a deep understanding of the changes, such as change propagation [30], defect detection [16], and code review of production code [15], [31].

Despite this success, studies on CIA in build systems have been limited. While prior research has explored the effects of changes to source or test code on build systems [32], [33] or their output, such as targets [34], [35], the impact of modified build specifications on the build system itself remains unexplored. Perhaps the most similar work to ours is that of Al-Kofahi et al. [36] where their tool, MkDiff, detects semantic changes in Makefiles. However, because MkDiff abstracts the build specifications into a Symbolic Dependency Graph (SDG) [18], it may obscure the logical pathways through which changes impact build rules, limiting its applicability for code review and debugging.

We propose BCIA, an approach that provides a clear view of how changes transitively affect the entire build system. BCIA automatically performs a global analysis on the build system, which helps to uncover a more comprehensive impact of changes and alleviates the need for such manual effort.

III. Research Hypothesis

In this thesis, we aim to understand how code review is applied to changes in build systems and improve this process by proposing an applicable and effective approach. Therefore, we aim to investigate Research Hypothesis.

To do so, we study the code review practices applied to changes in build systems and the purposes code review serves in this context. We also identify social and technical challenges that impede a thorough review of changes in build specifications. We propose BCIA to facilitate understanding of the implications changes may have on the build configuration space. Next, we evaluate the applicability of BCIA in an empirical study and its effectiveness in a user study, using BuiScout, a prototype for BCIA we implement to analyze CMake-based build systems. The remainder of this paper reports on our current progress and future plans towards the completion of this thesis.

IV. Understanding the Code Review of Changes in Build Systems

Motivation. Prior work highlights the importance of sufficient quality assurance for build systems [1]. However, automated quality assurance practices are often limited in or inapplicable to the context of build systems [8], [21], [22]. Among its many other benefits, code review improves code stability and software quality [10]. However, research shows that this well-adopted practice is not implemented for test code as rigorously as production code [12]. We suspect that this issue may also extend to build specifications. To investigate, we set out to answer two main research questions: **(RQ1) How is code review applied to changes in build systems?**, and **(RQ2) What are the challenges that impede a thorough code review of build systems?**

Approach. We quantitatively analyze 502,931 change sets from the large and active Qt and Eclipse communities. We compute popular measures of review intensity [37] for build systems and compare them to those of production and test code in various settings. Moreover, we qualitatively analyze 500 change sets to understand the purposes code review serves for build systems and the build-related concerns developers raise when reviewing changes. Lastly, we interview nine developers with 1-40 years of experience to understand the challenges that hinder a more rigorous review of build systems.

Results. Our quantitative analysis shows that changes to build systems are at least two times less likely to receive comments during code review when compared to production and test code, even when the change is solely focused on the build specifications. Our qualitative analysis reveals that code reviews in build systems are more likely to focus on defect detection than what is reported in the literature for reviews of production [10] and test code [12]. Furthermore, based on our qualitative analysis, we construct a taxonomy of build-specific issue patterns raised during code review of build systems. The results show that evolvability and dependency-related issues are the most frequently raised patterns of issues. Our interviewees point out social and technical factors that hinder a rigorous review of build specifications. While a prevalent lack of understanding of and interest in build systems among developers seems to be a major social challenge, our interviewees with more expertise in build systems, often tasked with build maintenance, hope for dedicated tooling to support the code review of build specifications. These results have been published in the proceedings of the International Conference on Software Engineering [13].

V. Improving the Code Review of Changes in Build Systems

Motivation. Maintenance and quality assurance of build systems imposes a considerable overhead on software development [1] and is a cognitively demanding task for developers [14]. The prevalent lack of interest and expertise in build systems [13], together with the complexities of build systems, may lead to developers dismissing seemingly small changes in build systems as trivial. However, similar to source code [38], such changes may have a far-reaching impact on the build configuration and output. Research shows that assisting reviewers in understanding the impact of changes through accurate CIA techniques improves the effectiveness and efficiency of the code review process in the context of production code [15], [16]. We reckon that CIA for changes in build specifications, i.e., Build Change Impact Analysis (BCIA), can achieve the same success in code review of changes in build systems.

Proposed Solution. To summarize the impact of changes to build specifications across the build configuration space, we propose Build Change Impact Analysis (BCIA). BCIA first traverses the data and control flow paths from before and after a change is applied to the build system and computes Conditional Definition-Use (CDU) chains [39] in a static analysis of the build specifications. BCIA then generates an Impact Knowledge Graph (IKG)—a knowledge graph that represents the impact of the change within and across CDU chains throughout the build system. To realize our BCIA, we produce BuiScout—a prototype implementation of BCIA for build systems that are written using CMake. Our prototype reveals the areas in the build system that are influenced by a change, which might not be readily apparent to developers when making or reviewing a change.

A. Applicability of Build Change Impact Analysis

Motivation. As build specifications are inherently different from source code (e.g., build code is often declarative rather than imperative in nature), it is not clear whether changes in build specifications propagate any impact to unmodified parts of the build system. As such, BCIA may prove to be inapplicable to changes in build systems. Therefore, we aim to answer the following research question: **(RQ3) Is change impact analysis applicable to the changes in build systems?**

Approach. To investigate the applicability of BCIA, we use BuiScout to conduct an empirical study of 10,000 change sets that span 10 large projects that have used CMake for a total of 28 years. In this empirical study, we measure the prevalence of change sets that propagate their impact to unmodified parts of the build system. Next, we characterize the impact these change sets propagate in terms of magnitude and breadth as heuristics to demonstrate that tracing them manually is not an insignificant effort.

Results. We find that in 93% of the change sets, the modifications exhibit characteristics that indicate an impact may propagate to areas of the build system beyond the modified locations. As such, a CIA is required to confirm the impact of the changes. BuiScout detects an impact on unmodified build configurations in over 77% of the change sets. Furthermore, these changes affect a median of 14 unmodified commands, out of which a median of 95.5% are non-local to the change set, i.e., they reside in unmodified build files.

B. Effectiveness of Build Change Impact Analysis

Motivation. Our findings in [17] suggest that changes in build systems may have an impact beyond the change set, and tracing the impact may involve investigating unmodified files. Our study of code review for build systems revealed that

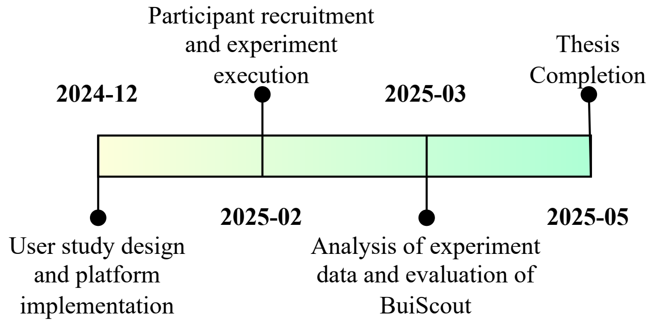

Participant recruitment and experiment execution
Thesis Completion
2024-12
2025-03
2025-02
2025-05
User study design and platform implementation
Analysis of experiment data and evaluation of BuiScout

Fig. 2: An overview of our timeline for the thesis completion.

build system maintainers suffer from this cognitive burden and require dedicated tool support to reduce this challenge [13]. We conjecture that dedicated approaches—such as BCIA—can alleviate the cognitive load developers endure when trying to grasp the full impact of changes in build systems, a key step in code review. We test this idea by answering the following research question: **(RQ4) Can change impact analysis improve the code review process for changes in build systems?**

Approach. We plan to answer this research question in a controlled within-subject user study. In this user study, the participants are asked to complete a series of tasks that involve identifying the impact of change sets. Each participant completes each task twice: once as a control without assistance from BuiScout, and once, on a different change set, as a treatment where they receive support through reports generated by BuiScout. We assess the accuracy and speed of impact localization as indicators of BCIA's effectiveness. Additionally, we explore whether using BuiScout reduces the perceived cognitive load associated with understanding the impact of build system changes. Figure 2 provides our timeline for the completion of this thesis.

VI. Conclusion

Due to the lack of readily deployable automated quality assurance techniques tailored for build systems, code review plays a key role in sustaining the quality of build systems. In this thesis, we establish that as important as code review is for sustaining the quality of build systems, it is not rigorously implemented into the build system maintenance process. Guided by the challenges that practitioners point out, we propose Build Change Impact Analysis (BCIA) as a dedicated approach to assist developers in understanding the impact of changes in build systems. We then demonstrate the applicability and effectiveness of BCIA in the context of code review for build systems.

This thesis provides insights into the current state of code review practices for Build system changes and presents actionable guidelines for managing the challenges that impede this process. It also proposes an applicable method to assist developers in understanding the impact of the changes in build systems. Our future plan demonstrates the effectiveness of our approach in real-life code review scenarios.

References

[1] S. McIntosh, B. Adams, T. H. Nguyen, Y. Kamei, and A. E. Hassan, "An Empirical Study of Build Maintenance Effort," in *the Proceedings of the International Conference on Software Engineering (ICSE)*, 2011, pp. 141–150.

[2] S. McIntosh, M. Nagappan, B. Adams, A. Mockus, and A. E. Hassan, "A Large-Scale Empirical Study of the Relationship between Build Technology and Build Maintenance," *Empirical Software Engineering (EMSE)*, vol. 20, no. 6, pp. 1587–1633, 2015.

[3] N. Nagappan and T. Ball, "Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study," in *the Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2007, pp. 364–373.

[4] Y. Zhang, Y. Jiang, C. Xu, X. Ma, and P. Yu, "ABC: Accelerated Building of C/C++ Projects," in *the Proceedings of the Asia-Pacific Software Engineering Conference (APSEC)*, 2015, pp. 182–189.

[5] H. Seo, C. Sadowski, S. Elbaum, E. Aftandilian, and R. Bowdidge, "Programmers' Build Errors: A Case Study (at Google)," in *the Proceedings of the International Conference on Software Engineering (ICSE)*, 2014, pp. 724–734.

[6] S. Nadi and R. Holt, "Make It or Break It: Mining Anomalies from Linux Kbuild," in *the Proceedings of the Working Conference on Reverse Engineering (WCRE)*, 2011, pp. 315–324.

[7] C. Macho, F. Oraze, and M. Pinzger, "DValidator: An Approach for Validating Dependencies in Build Configurations," *Journal of Systems and Software (JSS)*, vol. 209, p. 111916, 2024.

[8] T. Sotiropoulos, S. Chaliasos, D. Mitropoulos, and D. Spinellis, "A Model for Detecting Faults in Build Specifications," *the Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–30, 2020.

[9] C.-P. Bezemer, S. McIntosh, B. Adams, D. M. German, and A. E. Hassan, "An Empirical Study of Unspecified Dependencies in Make-based Build Systems," *Empirical Software Engineering (EMSE)*, vol. 22, no. 6, pp. 3117–3148, 2017.

[10] A. Bacchelli and C. Bird, "Expectations, Outcomes, and Challenges of Modern Code Review," in *the Proceedings of the International Conference on Software Engineering (ICSE)*, 2013, pp. 712–721.

[11] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating Code Review Quality: Do People and Participation Matter?" in *the Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 111–120.

[12] D. Spadini, M. Aniche, M.-A. Storey, M. Bruntink, and A. Bacchelli, "When Testing Meets Code Review: Why and How Developers Review Tests," in *the Proceedings of the International Conference on Software Engineering (ICSE)*, 2018, pp. 677–687.

[13] M. Nejati, M. Alfadel, and S. McIntosh, "Code Review of Build System Specifications: Prevalence, Purposes, Patterns, and Perceptions," in *the Proceedings of the International Conference on Software Engineering (ICSE)*, 2023, pp. 1213–1224.

[14] S. Phillips, T. Zimmermann, and C. Bird, "Understanding and Improving Software Build Teams," in *the Proceedings of the International Conference on Software Engineering (ICSE)*, 2014, pp. 735–744.

[15] Q. Hanam, A. Mesbah, and R. Holmes, "Aiding Code Change Understanding with Semantic Change Impact Analysis," in *the Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 202–212.

[16] S. Jiang, C. McMillan, and R. Santelices, "Do Programmers Do Change Impact Analysis in Debugging?" *Empirical Software Engineering (EMSE)*, no. 2, pp. 631–669, 2017.

[17] M. Nejati, M. Alfadel, and S. McIntosh, "Understanding the Implications of Changes to Build Systems," in *the Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2024.

[18] A. Tamrawi, H. A. Nguyen, H. V. Nguyen, and T. N. Nguyen, "Build Code Analysis with Symbolic Evaluation," in *the Proceedings of the International Conference on Software Engineering (ICSE)*, 2012, pp. 650–660.

[19] C. Macho, S. McIntosh, and M. Pinzger, "Automatically Repairing Dependency-related Build Breakage," in *the Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 106–117.

[20] F. Hassan and X. Wang, "Hirebuild: An Automatic Approach to History-Driven Repair of Build Scripts," in *the Proceedings of the International Conference on Software Engineering (ICSE)*, 2018, pp. 1078–1089.

[21] Y. Lou, J. Chen, L. Zhang, D. Hao, and L. Zhang, "History-Driven Build Failure Fixing: How Far Are We?" in *the Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, 2019, pp. 43–54.

[22] G. Fan, C. Wang, R. Wu, X. Xiao, Q. Shi, and C. Zhang, "Escaping Dependency Hell: Finding Build Dependency Errors with the Unified Dependency Graph," in *the Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, 2020, pp. 463–474.

[23] C. Lebeuf, E. Voyloshnikova, K. Herzig, and M.-A. Storey, "Understanding, Debugging, and Optimizing Distributed Software Builds: A Design Study," in *the Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 496–507.

[24] R. Hardt and E. V. Munson, "An Empirical Evaluation of Ant Build Maintenance Using Formiga," in *the Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 201–210.

[25] C. Vassallo, S. Proksch, T. Zemp, and H. C. Gall, "Every Build You Break: Developer-Oriented Assistance for Build Failure Resolution," *Empirical Software Engineering (EMSE)*, vol. 25, no. 3, pp. 2218–2257, 2020.

[26] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern Code Reviews in Open-source Projects: Which Problems Do They Fix?" in *the Proceedings of the Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 202–211.

[27] M. V. Mäntylä and C. Lassenius, "What Types of Defects are Really Discovered in Code Reviews?" *IEEE Transactions on Software Engineering (TSE)*, vol. 35, no. 3, pp. 430–448, 2008.

[28] S. Panichella, V. Arnaoudova, M. Di Penta, and G. Antoniol, "Would Static Analysis Tools Help Developers with Code Reviews?" in *the Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 161–170.

[29] Bohner, "Impact analysis in the software change process: a year 2000 perspective," in *the Proceedings of the International Conference on Software Maintenance (ICSM)*, 1996, pp. 42–51.

[30] Z. Jiang, H. Zhong, and N. Meng, "Investigating and Recommending Co-Changed Entities for JavaScript Programs," *Journal of Systems and Software (JSS)*, vol. 180, p. 111027, 2021.

[31] Y. Huang, N. Jia, X. Chen, K. Hong, and Z. Zheng, "Salient-Class Location: Help Developers Understand Code Change in Code Review," in *the Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2018, pp. 770–774.

[32] C. Macho, S. McIntosh, and M. Pinzger, "Predicting Build Co-changes with Source Code Change and Commit Categories," in *the Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, pp. 541–551.

[33] X. Xia, D. Lo, S. McIntosh, E. Shihab, and A. E. Hassan, "Cross-Project Build Co-Change Prediction," in *the Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 311–320.

[34] R. Wen, D. Gilbert, M. G. Roche, and S. McIntosh, "BLIMP Tracer: Integrating Build Impact Analysis with Code Review," in *the Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 685–694.

[35] M. Meidani, M. Lamothe, and S. McIntosh, "Assessing the Exposure of Software Changes: The DiPiDi Approach," *Empirical Software Engineering (EMSE)*, vol. 28, no. 2, p. 41, 2023.

[36] J. M. Al-Kofahi, H. V. Nguyen, A. T. Nguyen, T. T. Nguyen, and T. N. Nguyen, "Detecting Semantic Changes in Makefile Build Code," in *The Proceedings of the International Conference on Software Maintenance (ICSM)*, 2012, pp. 150–159.

[37] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Review Participation in Modern Code Review," *Empirical Software Engineering (EMSE)*, vol. 22, no. 2, pp. 768–817, 2017.

[38] P. Dai, Y. Wang, D. Jin, Y. Gong, and W. Yang, "An improving approach to analyzing change impact of C programs," *Journal of Computer Communications (ComputCommun)*, vol. 182, pp. 60–71, 2022.

[39] N. Parasaram, E. T. Barr, and S. Mechtaev, "Rete: Learning Namespace Representation for Program Repair," in *the Proceedings of the International Conference on Software Engineering (ICSE)*, 2023, pp. 1264–1276.