# Towards Efficient and Secure Pull-Request-Based Software Development

Kazi Amit Hasan
kaziamit.hasan@queensu.ca
Queen's University
Kingston, Ontario, Canada

## Abstract

Pull-request-based development (PBD) is the dominant workflow in modern open-source software (OSS), where pull requests (PRs) are the central coordination unit for review, CI/CD validation, and fixing security vulnerabilities. Existing research studies PR responsiveness, CI/CD performance, and vulnerability handling in isolation, even though delays and security risks accumulate through the same PR-mediated workflow. To address this gap, this thesis treats PBD as a unified workflow and investigates how suboptimal PR management creates avoidable PR delays and increased security risk. We provide empirical evidence and tools to address these coupled outcomes. Overall, the thesis connects efficiency and security through one PR-based workflow. It provides actionable guidance for reducing PR delays by characterizing and improving key efficiency bottlenecks, and offers recommendations for improving vulnerability management by developing an LLM-based, low-noise, evidence-driven security alert system.

## Keywords

PR management, Vulnerability management, Software supply chain security

**ACM Reference Format:**
Kazi Amit Hasan. 2026. Towards Efficient and Secure Pull-Request-Based Software Development. In *Companion Proceedings of the 34th ACM Symposium on the Foundations of Software Engineering (FSE '26), June 5–9, 2026, Montreal, Canada.* ACM, New York, NY, USA, 5 pages.

## 1 Introduction

PR-Based development (PBD) is widely adopted in modern open-source software (OSS) projects, where developers propose code changes by submitting a pull request (PR) that is then discussed, reviewed, validated by CI/CD, and eventually merged [1]. In practice, PRs become the *central coordination unit* for both engineering decisions (review, testing, integration) and security actions (building security fix patches and preparing releases) [2, 3].

**Definitions:** PR-based development (PBD) is a pull-based workflow where developers propose changes through pull requests (PRs), and those PRs serve as the main unit for review, CI validation, and integration into the codebase [1]. In this thesis, we use *pull request management* to refer how developers coordinate and manage current practices and how mismanagement within the workflow lead to avoidable PR delays and elevated security risks [1, 4].

**Motivation:** When a PR management is suboptimal, projects incur two coupled costs. First, the development efficiency gets affected. Varying first response developers, use of automated tools, ineffective caching practices often delay reviews, prolong PR lifetime and increase contributor retentions [2, 5]. The same delay along with developer's practices also affect security fixes, which are proposed and fixed through PRs and later get released for other dependent projects. As a result, the subsequent releases are delayed, which can extend the downstream security risks [6]. Motivated by this, our thesis studies suboptimal PR management as one problem with two coupled outcomes which are avoidable PR delays and avoidable downstream security risk.

**Challenges:** Three recurring challenges motivate our work.

**(C1) First response in PBD:** The time to the first response is a widely used indicator of how responsive project maintainers are [2]. However, modern PR workflows frequently include automated bots that comment, label, and run checks [7]. These responses can make a project appear responsive even when the first *human* response is delayed. This motivates the need to understand the time to first response and how it affects the PR review process.

**(C2) CI/CD efficiency in PBD:** PBD also gets affected when CI/CD workflows are slow and repeatedly re-execute redundant work [5]. Caching is a common mitigation, but it is not a one-time optimization. It must be configured, updated, debugged, and sometimes reverted, and ineffective caching can prolong the PR lifecycle [8]. While prior work studies CI/CD evolution holistically [9], caching is treated as one of many general maintenance activities. Our work presents the first cache-centric empirical study of GitHub Actions workflows.

**(C3) Vulnerability management in PBD:** Vulnerability management in PBD is often non-coordinated. Upstream projects vary in how they disclose and document security fixes [3], while downstream projects rely on automated dependency-update tools that can be noisy and inconsistently configured [10]. Even with such tools, adoption delays frequently remain substantial [11].

**Gaps addressed in this thesis:** This thesis treats PBD as a unified PR workflow, namely, the end to end sequence of PR-mediated steps through which (i) a change is proposed, (ii) reviewers and maintainers respond and iterate, (iii) CI/CD validates revisions, (iv) the change is merged, and (v) when the change is security-relevant, it is released and then adopted downstream. Under this view, efficiency and security are inherently linked through the same PR lifecycle and examined through the lens of PR management, but together they provide an end-to-end view of how suboptimal PR management systematically accumulates delays and security risk across the PR-based workflow. We test the following hypothesis:

***Research Hypothesis:*** *Suboptimal PR management, characterized by inconsistent first-response timeliness and ineffective caching practices, combining with non-coordinated vulnerability management, lead to substantial and avoidable PR delays and elevated security risk in PR based development.*

To test this hypothesis, we conduct two connected studies aligned with Figure 1. First (PS1, completed), we examine efficiency bottlenecks in PDB: **RQ1** distinguishes bot-first from human-first responses and shows that actionable human responsiveness better explains PR outcomes, while **RQ2** analyzes efficiency-oriented mechanisms in GitHub Actions, revealing that caching is job-specific and that failure-driven changes can prolong CI feedback cycles. These findings support the efficiency dimension of our hypothesis. Second (PS2, in progress), we study vulnerability management: **RQ3** quantifies adoption delays and shows that non-coordinated practices and automated dependency tools do not ensure timely updates, and **RQ4** proposes an LLM-based system that provides early, evidence-backed, low-noise security signals to downstream projects to reduce the risk window. Overall, this thesis connects efficiency and security through PR-based workflows and provides empirical evidence and practical tool directions to improve both efficiency and security in PDB.

## 2 Understanding and improving efficiency bottlenecks in PR-based development

PBD depends on timely reviewer feedback and efficient CI validation, yet PRs often stall due to delayed first responses and redundant workflow executions. Although bots and CI/CD aim to reduce these delays, bot-first replies can obscure actionable responsiveness, and efficiency-oriented mechanisms such as caching can introduce overhead when configured or maintained sub-optimally. We hypothesize that *inconsistent first-response timeliness* and *ineffective caching practices* are core contributors to suboptimal PR management. To examine this, we aim to answer two research questions: **(RQ1) How does time-to-first-response affect PBD?** and **(RQ2) What challenges in existing efficiency-oriented mechanisms affect PBD?**

### 2.1 (RQ1) Characterizing time to first response

**Motivation:** PRs frequently stall while waiting for the first response, which can reduce development efficiency and newcomer retention. Since our thesis hypothesizes that *inconsistent first-response timeliness* is a core symptom of suboptimal PR management, we study time-to-first-response as an actionable factor for reducing avoidable PR delays.

**Approach:** We first conduct a preliminary study leveraging the most recent and largest PR latency benchmark dataset provided by Zhang et al. [2]. We observe that the time-to-first-response should be carefully interpreted, as many PRs with extremely short time-to-first-response are generated by bots. Motivated by this observation, we propose to redefine this concept by considering whether the response is generated by a human or bot. Then, we conduct an in-depth empirical study of various types of time-to-first-response in ten popular and mature projects and classify bots and human responders using BoDeGha[1], and validated our approach.

**Results:** Bots are present in all target projects and are responsible for up to 97.08% first responses, but bot-first response time explains little of PR lifetime (0.33% variance). In contrast, the time to the subsequent first *human* response explains substantially more (64.70%). We also find that 36% of PRs with long first-response times have short review processes, suggesting potential lifetime reductions if maintainers respond earlier. We also find that PR complexity (e.g., lengthy descriptions) and contributor inexperience correlate with longer waits for the first human response, and newcomer retention is 2.63%–37.44% higher when the first human response arrives within one day [12]. Overall, these results indicate that delayed first-human responses are a key workflow bottleneck linking suboptimal PR management to avoidable PR delays.

### 2.2 (RQ2) Characterizing usage patterns and identifying challenges

**Motivation:** In PBD, CI/CD pipelines automatically validate code changes submitted through PRs, often triggering repeated executions of build, dependency installation, and test workflows, which introduces substantial redundancy across runs [13]. To reduce these costs, CI/CD pipelines rely on efficiency-oriented mechanisms, such as **caching**, to reuse previously computed artifacts. However, when caching is configured and maintained sub-optimally, its effectiveness degrades, prolonging CI feedback cycles and contributing to PR stalling. Since our thesis hypothesizes that *ineffective caching practices* are a core symptom of suboptimal PR management, we examine how caching is adopted and evolves over time, and the challenges that limit its efficiency.

**Approach:** Our study builds on the dataset of Bouzenia et al. [5], comprising 952 GitHub repositories and 1.3 million CI workflow executions. Among them, we only focused on *caching* adopting repositories (27.9%). We collected repository-level metadata for all projects via the GitHub GraphQL (v4) API[2] and reconstructed complete workflow histories of caching adopters using *Gigawork*.[3] Workflow jobs were categorized using developer-defined job names, adapting the CI/CD phase taxonomy of Bouzenia et al. [5]. Unlike prior work that primarily focuses on *actions/cache* [14], we broaden our scope using GitHub documentation[4] to capture both explicit and implicit caching enabled by default. We retained only commits modifying caching configurations to track how caching is introduced, modified, and removed across CI/CD phases, and complemented the quantitative analysis with a qualitative study of caching-related commits through diffs, commit messages, and linked PR discussions.

**Results:** Our findings show that caching-adopting repositories tend to be larger and more active, and strategically applied in resource-intensive CI/CD phases, through direct and package-manager caching. Cache evolution is highly phase-specific. Build and test jobs exhibit rapid, iterative parameter and version tuning, while release and integration jobs follow slower, cautious update cycles. Parameter updates are failure-driven and frequent (median 4.29 days), whereas version updates are rare (median 384.13 days). We also observe rapid cache removals due to ineffective optimization and cache downgrades where humans manually recover from failed automated

[1] https://github.com/sgl-umons/BoDeGHa

[2] https://docs.github.com/en/graphql

[3] https://github.com/cardoeng/gigawork

[4] https://github.com/actions/cache

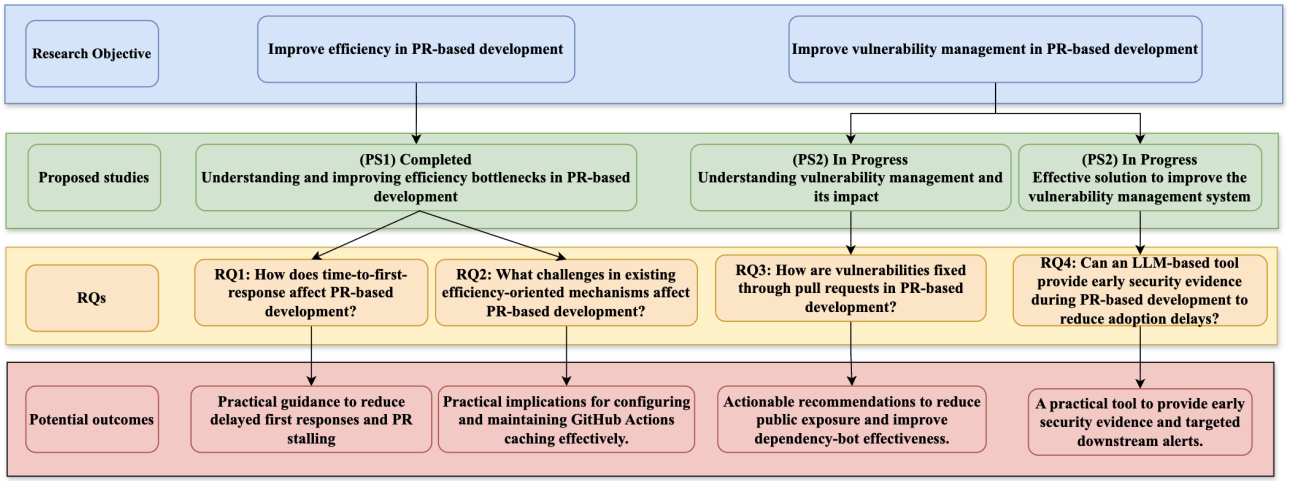


**Figure 1: An overview of our research thesis.**

upgrades. Overall, caching requires continuous maintenance, and failure-driven fixes can prolong CI latency and contribute to avoidable PR delays, supporting our thesis hypothesis.

## 3 Understanding and improving vulnerability management system

PBD is also the backbone of how vulnerabilities are fixed and propagated across the software supply chain. Security patches are typically created, reviewed, and merged through PRs and only later reach downstream projects through releases. However, upstream fixes do not immediately reduce ecosystem risk, as exposure persists while downstream projects continue using vulnerable versions. This process is further hindered by inconsistent vulnerability management practices and the limited effectiveness of existing tools, which are often noisy, inconsistently configured, and lack actionable security context. Motivated by this gap, our thesis examines how vulnerabilities are managed and resolved in PBD, and how adoption delays can be reduced across the dependency chain. We address this theme through two research questions: **(RQ3) How are vulnerabilities fixed through PRs in PBD?** and **(RQ4) Can an LLM-based tool provide early security evidence during PBD to reduce adoption delays?**

### 3.1 (RQ3) Prevalence of non-coordinated vulnerability management practices

**Motivation:** In PBD, vulnerability fixes are delivered through PR-mediated actions, but upstream patch availability does not immediately reduce risk in the dependency chain. Downstream projects often continue using vulnerable versions due to manual and irregular dependency upgrades, creating adoption delays during which known vulnerabilities can be exploited. While prior work focuses on upstream vulnerability management practices, we lack a systematic understanding of how vulnerability management in upstream PBD affects downstream adoption and how effectively the existing tools mitigate these delays. This thesis quantifies adoption delays and examines how tool adoption patterns shape the downstream risk window in PBD.

**Approach:** We use the dataset of Wu et al. [15], comprising 832 CVEs, 1,078 fixes across 613 upstream projects and their downstream dependents. We classify fixes as *Silent* or *Transparent* based on reporting visibility (public/private) and the presence of security policies, and manually analyze all security policies using open coding to derive pragmatic recommendations. We enrich the dataset with PR/issue creation dates, security release dates, and NVD disclosure dates to study risk transfer from upstream to downstream projects. Finally, we quantify downstream adoption delays and analyze dependency management tools and their configurations to assess their effectiveness in reducing these delays.

**Obtained Results:** Our results show that developers frequently do not follow maintainers' security policies, even when private reporting and silent fix guidelines are provided. We found 67.70% of projects have security policies while other do not have them. Our analysis identifies redundant and ambiguous instructions and derive implications for maintainers. 58.4% of the downstream projects delay updates by over three months, and only 1.7% update on the same day as the upstream fix. Our analysis confirms the presence of Dependabot in downstream projects. Despite tool adoption, downstream delays persist, motivating further investigation into the effectiveness of Dependabot. Currently, we are working on understanding and quantifying the effectiveness of dependabot. Overall, these findings highlight opportunities to improve vulnerability management in PBD through stronger upstream reporting and more effective dependency-management tools.

### 3.2 (RQ4) Improving vulnerability management system

**Motivation:** Building on **RQ3**, we find that automated dependency management tools do not reliably reduce downstream adoption delays in PBD. While tools such as Dependabot generate security update PRs, these are often perceived as noisy and overwhelming

[16]. Moreover, such tools provide limited context about vulnerability severity, impact, or consequences of delayed adoption. To address this, we therefore propose an LLM-based system that extracts early security evidence from upstream PR workflows and delivers targeted, low-noise alerts to downstream projects. Our hypothesis is that early, evidence-backed, project-specific signals can address these limitations and reduce adoption delays.

**Approach:** We plan to answer **RQ4**, by proposing a multi-agent system that analyzes upstream PR activity to extract *early, evidence-backed* security signals and translate them into actionable downstream guidance. The system prioritizes and filters security alerts to reduce noise by adding contextual information and generating concise recommendations for impacted downstream projects. We will evaluate the approach by measuring its ability to reduce alert noise and shorten downstream adoption delays compared to existing automated dependency update workflows

**Expected Results:** We expect three outcomes. First, we anticipate our system will produce high-precision, evidence-backed security signals to help maintainers recognizing severity based security updates without extensive manual investigation. Second, we expect reduced noise compared to existing dependency management tools by prioritizing only high-confidence and high-impact cases and by providing concise, actionable summaries. Third, we expect measurable improvements in downstream responsiveness, including shorter adoption delays for security fixes. Together, these outcomes support our thesis goal of improving vulnerability management in PBD through practical, deployable automation that complements existing dependency management tools.

## 4 Related Work

**Code change management in PBD:** Prior work characterizes the pull-based model on GitHub and factors associated with PR outcomes [1], and examines PR latency and decision-making [17]. Time-to-first-response is commonly used to capture reviewer responsiveness [2], but automated bots complicate its interpretation [18]. This motivates distinguishing bot-first from first-human responses, which we address in our thesis.

**CI/CD and caching practices in PBD:** CI/CD provides automated validation before merge in PBD. Recent studies examine CI/CD configurations and practices [13] and GitHub Actions' impact on PR processes [19], including adoption, evolution/maintenance, outdatedness, and developer pain points [9]. Caching reduces redundant CI work and can accelerate builds [8], but workflow maintenance and configuration drift can undermine these gains [9]. We connect these issues to PR outcomes by studying how caching practices and misconfigurations contribute to avoidable PR delays.

**Vulnerability management in PBD:** Vulnerability fixes are often developed via PRs and later reach downstream projects through releases. Prior work studies release and disclosure behaviors in OSS and quantifies lags [3]. To reduce adoption delays, projects increasingly rely on automated dependency-update tools such as Dependabot [5]. Prior empirical studies report the effectiveness of Dependabot in resolving vulnerabilities [10]. Despite of its effectiveness, developers find PRs generated by Dependabot are noisy [16]. We address this gap by developing an LLM-based system that resolve this and provide targeted and low-noise alerts to developers.

[5]https://github.com/dependabot

## 5 Conclusion

Despite the prevalence of pull-request-based development (PBD), responsiveness, CI/CD efficiency, and vulnerability management are often addressed in isolation, even though delays and security risk compound through the same PR-mediated workflow. This thesis treats PBD as a unified workflow and argues that suboptimal *PR management* creates two coupled outcomes: avoidable PR delays and avoidable downstream security exposure. Our results identify workflow bottlenecks that prolong PR throughput and extend exposure windows after security fixes are released, motivating actionable recommendations and tooling directions to improve efficiency while reducing risk. Future work will evaluate an LLM-based, low-noise, evidence-driven security alert system that extracts early security signals from upstream PR activity to support faster adoption.

## References


[1] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: the contributor's perspective," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 285–296.

[2] X. Zhang, Y. Yu, T. Wang, A. Rastogi, and H. Wang, "Pull request latency explained: An empirical overview," *Empirical Software Engineering*, vol. 27, no. 6, pp. 1–38, 2022.

[3] N. Imtiaz, A. Khanom, and L. Williams, "Open or sneaky? fast or slow? light or heavy?: Investigating security releases of open source packages," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1540–1560, 2022.

[4] C. Maddila, S. S. Upadrasta, C. Bansal, N. Nagappan, G. Gousios, and A. v. Deursen, "Nudge: Accelerating overdue pull requests towards completion," *ACM Trans. Softw. Eng. Methodol.*, June 2022.

[5] I. Bouzenia and M. Pradel, "Resource usage and optimization opportunities in workflows of github actions," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–12.

[6] B. Chinthanet, R. G. Kula, S. McIntosh, T. Ishio, A. Ihara, and K. Matsumoto, "Lags in the release, adoption, and propagation of npm vulnerability fixes," *Empirical Software Engineering*, vol. 26, pp. 1–28, 2021.

[7] M. Wessel, A. Serebrenik, I. Wiese, I. Steinmacher, and M. A. Gerosa, "Effects of adopting code review bots on pull requests to oss projects," in *2020 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2020, pp. 1–11.

[8] K. Gallaba, J. Ewart, Y. Junqueira, and S. McIntosh, "Accelerating continuous integration by caching environments and inferring dependencies," *IEEE Transactions on Software Engineering*, vol. 48, no. 6, pp. 2040–2052, 2020.

[9] P. Valenzuela-Toledo, A. Bergel, T. Kehrer, and O. Nierstrasz, "The hidden costs of automation: An empirical study on github actions workflow maintenance," in *2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2024, pp. 213–223.

[10] M. Alfadel, D. E. Costa, E. Shihab, and M. Mkhallalati, "On the use of dependabot security pull requests," in *2021 IEEE/ACM 18th International conference on mining software repositories (MSR)*. IEEE, 2021, pp. 254–265.

[11] R. He, H. He, Y. Zhang, and M. Zhou, "Automating dependency updates in practice: An exploratory study on github dependabot," *IEEE Transactions on Software Engineering*, vol. 49, no. 8, pp. 4004–4022, 2023.

[12] K. A. Hasan, M. Macedo, Y. Tian, B. Adams, and S. Ding, "Understanding the time to first response in github pull requests," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 2023, pp. 1–11.

[13] T. Ghaleb, O. Abduljalil, and S. Hassan, "Ci/cd configuration practices in open-source android apps: An empirical study," *ACM Transactions on Software Engineering and Methodology*, 2024.

[14] Y. Zhang, Y. Wu, T. Chen, T. Wang, H. Liu, and H. Wang, "How do developers talk about github actions? evidence from online software development community," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.

[15] Y. Wu, Z. Yu, M. Wen, Q. Li, D. Zou, and H. Jin, "Understanding the threats of upstream vulnerabilities to downstream projects in the maven ecosystem," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1046–1058.

[16] C. Miller, H. He, W. Chen, E. Lin, C. Yang, B. Vasilescu, and C. Kästner, "Designing abandabot: When does open source dependency abandonment matter?" in *Proc.*

*Int'l Conf. Software Engineering (ICSE)*, 2026.
[17] X. Zhang, Y. Yu, G. Georgios, and A. Rastogi, "Pull request decisions explained: An empirical overview," *IEEE Transactions on Software Engineering*, 2022.
[18] Z. Wang, Y. Wang, and D. Redmiles, "From specialized mechanics to project butlers: the usage of bots in oss development," *IEEE Software*, 2022.
[19] M. Wessel, J. Vargovich, M. A. Gerosa, and C. Treude, "Github actions: the impact on the pull request process," *Empirical Software Engineering*, vol. 28, no. 6, p. 131, 2023.