Trustworthiness of Continuous Integration Practices*

Pablo Valenzuela-Toledo
University of Bern
Bern, Switzerland
Universidad de La Frontera
Temuco, Chile
pablo.valenzuela@unibe.ch

Abstract

Continuous Integration Practices (CIP) (*i.e.*, the socio-technical activities, and processes that govern how software changes are frequently integrated, built, and tested), are central to modern software development. While CIP provide rapid feedback and promote seamless collaboration, their growing scale and complexity introduce challenges in workflow maintenance, failure diagnosis, and security risk. These challenges can erode developers' confidence, indicating that trust in CIP depends not only on technical soundness but also on the ability to comprehend and control the processes. We refer to this overarching construct as the *trustworthiness of CIP*, which captures the extent to which developers can place justified confidence in automation despite its challenges and limitations.

In this thesis, we examine the *trustworthiness of CIP* as a means to explain the dynamics of confidence in these practices. We argue that overlooking trustworthiness concerns undermines developers' confidence and limits the effectiveness of CIP. We aim to quantify and characterize manifestations of untrustworthiness and to propose solutions that help developers sustain confidence in their CIP.

CCS Concepts

• **Software and its engineering → Continuous integration**; *Software visualization*; *Program comprehension*.

Keywords

Continuous Integration Practices, Trustworthiness, Automation

ACM Reference Format:
Pablo Valenzuela-Toledo. 2026. Trustworthiness of Continuous Integration Practices. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-Companion '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3774748.3787654

1 Introduction

Continuous Integration Practices (CIP) rest on the premise that committing small and frequent changes to shared repositories reduces integration problems and provides rapid feedback to developers [8, 12]. These practices are now standard in modern software engineering, linked to higher productivity through faster feedback and reduced integration risks [5, 7].

CIP centers on pipelines that automatically integrate code at each commit and record the outcomes of their executions. As projects evolve, these pipelines grow in complexity, raising concerns of maintenance, feedback, and security, each affecting a different facet of their trustworthiness [13]. At the system level, recurrent configuration changes and repairs undermine consistency and reliability [10, 18, 25]. From an interaction perspective, weak feedback loops limit developers' ability to diagnose failures and interpret pipelines' execution results [6, 27–29]. Regarding the control perspective, third-party components can expose pipelines to vulnerabilities and reduce trust in their integrity [1, 4, 11, 15].

We refer to this broader concern as the *trustworthiness of CIP*, which, in the context of software tools, refers to the extent to which developers can rely on their continuous integration processes [9, 13]. While prior research has explored technical aspects such as pipeline efficiency or resource waste, little attention has been devoted to understanding the trustworthiness in these practices [2, 26]. In this thesis, we examine factors that are particularly relevant within the context of CIP: how developers maintain, interpret failures, and secure their pipelines. We quantify and characterize these manifestations and propose solutions to improve the reliability of these practices. We state our thesis as follows.

Thesis Statement

Enhancing the trustworthiness of CIP requires a holistic view of how developers maintain, interpret, and secure their pipelines. This perspective can be pursued through the analysis of three dimensions: (1) its evolution and maintenance, (2) the comprehension and diagnosis of failures, and (3) the detection and mitigation of security issues.

2 Research Scope and Objectives

The scope of this thesis covers three dimensions of CIP trustworthiness: maintenance, failure diagnosis, and security (Figure 1). Each dimension is examined through empirical studies combining qualitative and quantitative methods. The findings provide practical guidance for strengthening trust in CIP and form the basis for tools that support maintenance analysis, execution understanding, and security assessment. This research is conducted during the fourth year of a five-year PhD program.

2.1 CIP Maintenance

As projects evolve, CIP pipelines undergo continuous changes, which can increase maintenance overhead and reduce reliability. To examine this phenomenon, we mined 183 GitHub repositories using GitHub

*Advisors: Prof. Dr. Timo Kehrer & Dr. Sebastiano Panichella.

ICSE-Companion '26, April 12–18, 2026, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2296-7/2026/04
https://doi.org/10.1145/3774748.3787654

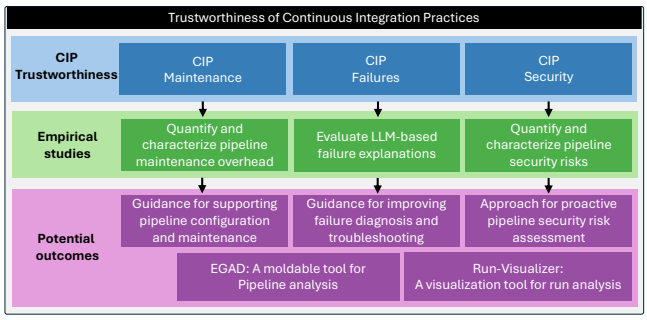

Figure 1: Research Methodology.

Actions (GA) across ten programming languages and extracted full commit-level data. We normalized file histories by category (production, test, pipeline), computed churn and coupling metrics (support, confidence, lift), and validated distributional differences through non-parametric statistics. Our findings show that pipelines constitute 3.7% of project files, yet their mean size is comparable to that of test code and their monthly churn rate reaches 8.2%, confirming that they evolve continuously despite their limited proportion among project files. These changes frequently co-occur with production and test edits associated with bug fixing, CI/CD improvement, and dependency management. They reflect reactive maintenance patterns concentrated among a small subset of contributors, increasing the likelihood of knowledge silos. A full version of this study appears in our published work [23].

Findings from complementary studies reinforce this view. We qualitatively inspected 222 pipeline commits, identifying frequent debugging and YAML syntax fixes aimed at restoring execution consistency and ensuring correct configuration. As a consequence of these findings, we developed *EGAD*, an analytical tool that reifies pipelines as first-class entities and enables the identification of instability patterns such as sticky commits. Full discussion is presented in our published work [20–22]. As these issues stem from unclear documentation and limited feedback, our ongoing research analyzes Stack Overflow discussions to identify GA developers' information needs. We identify recurring needs that reveal deficiencies in documentation and tooling, and plan to submit the results in late 2025.

2.2 CIP Failures

Run (*i.e.*, pipeline execution) failures disrupt automation, waste computational resources, and may thereby undermine developers' confidence. Such failures often stem from misconfigurations, dependency issues, or unstable environments, while the large and unstructured nature of logs makes their diagnosis demanding and time-consuming. These challenges motivated us to investigate whether recent advances in large language models (LLMs), which have proven effective in code understanding and summarization tasks, could aid in interpreting unstructured logs and supporting the diagnosis of CI/CD failures. To this end, we developed *LogExp*, a web-based tool that displays execution logs alongside LLM-generated explanations. In a developer survey, more than 80% of participants rated these explanations as correct and clear for simpler logs, although their usefulness decreased for complex failures involving multiple components. Developers appreciated concise, context-aware summaries but emphasized the need for richer contextual integration to make explanations actionable. A detailed discussion of this study is available in our published paper [24].

Building on these findings, we developed *Run-Visualizer*, a tool that reifies workflow runs as first-class entities to provide full context. It models each run's structural and behavioral context, enabling developers to identify where and how failures occur. The tool maps runs to visual narratives that facilitate exploration across multiple granularities. We plan to conduct a user study to evaluate how this representation supports diagnosis and corrective actions during run analysis, with submission targeted for early 2026.

2.3 CIP Security

CIP pipelines often rely on third-party components (*e.g.*, GitHub Actions) that may introduce security flaws and compromise their integrity. Misconfigured permissions, unsafe secret management, and unverified Actions increase the attack surface, while current safeguards mainly operate after adoption, providing limited visibility into the actual security risk. To address these limitations, we introduced the notion of *vulnerability-proneness*, defined as the likelihood that an Action contains security flaws, and used it as a pre-adoption criterion for risk assessment. We analyzed 270 Marketplace Actions to evaluate whether their metadata can indicate security risks. Results show that these metadata attributes, such as verification or popularity, provide limited and inconsistent information for indicating security risk, and that even Actions with favorable metadata profiles may still contain severe vulnerabilities. In contrast, repository-level metadata provide stronger indicators of risk. Building on this findings, we developed lightweight classification models that predict vulnerability-proneness using only repository metadata (*e.g.*, number of contributors), achieving an F1 score of 0.745 and a ROC AUC of 0.856. This study is currently submitted and under review.

Our previous findings show that security weaknesses in CIP pipelines persist despite ongoing maintenance and corrective efforts. Although pipelines undergo periodic updates to fix or replace vulnerable components, these attempts often achieve only partial effectiveness. To further investigate this issue, we mine workflow histories from open-source repositories and statically analyze each version to detect insecure configurations or outdated vulnerable Actions. Early results indicate that insecure patterns often persist or reemerge after partial remediation. The full study is planned for submission in mid-2026.

3 Related Work

Research on CIP has addressed the maintenance and evolution. Studies report that pipelines undergo frequent configuration edits that disrupt execution and require repeated repair [10, 25]. Other analyses describe redundant jobs, outdated configurations, migration challenges, missing optimization, bad smells, and brittle YAML structures as typical sources of instability [2, 3, 14, 18, 19].

Failure diagnosis has been examined as another critical aspect of CIP operation. Research shows that feedback mechanisms often lack clarity, forcing developers to interpret fragmented or ambiguous logs and to rely on trial and error or peer discussions to identify

faults [19]. Execution inefficiencies and underused caching further prolong diagnosis and delay recovery [2, 28].

Security-related studies focus on the integration of third-party components within pipelines. Audits of GitHub Actions identified excessive permissions, unverified sources, and unpinned dependencies that expose workflows to interference [1, 11, 15]. Static and taint analyses revealed injection flaws propagating through shared components [16, 17], while dependency analyses showed reliance on outdated or vulnerable packages [3, 4, 17].

Existing studies on CIP offer a view of how pipelines evolve, fail, and expose risks. However, they suggest that the reliability of CIP cannot be understood solely through technical aspects. Building on these observations, our thesis takes a broader perspective and, unlike prior work, frames these issues under the integrative construct of trustworthiness to explain how confidence in automation is formed and sustained.

4 Conclusion

This thesis examines the trustworthiness of CIP by analyzing how pipelines evolve, fail, and remain secure. The findings show that confidence in CIP depends not only on the technical soundness of automation but also on developers' capacity to comprehend and manage its behavior. Through empirical analyses, this research characterizes the factors that influence this confidence and introduces practical means to improve maintenance, diagnosis, and security across continuous integration practices.

Acknowledgments. The author would like to thank the Swiss (CH) Group for Original and Outside-the-box Software Engineering (CHOOSE) for providing financial support for the conference travel.

References

[1] Giacomo Benedetti, Luca Verderame, and Alessio Merlo. 2022. Automatic security assessment of GitHub actions workflows. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*. 37–45.

[2] Islem Bouzenia and Michael Pradel. 2024. Resource Usage and Optimization Opportunities in Workflows of GitHub Actions. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12.

[3] Alexandre Decan, Tom Mens, and Hassan Onsori Delicheh. 2023. On the outdatedness of workflows in the GitHub Actions ecosystem. *Journal of Systems and Software* 206 (2023), 111827.

[4] Hassan Onsori Delicheh, Alexandre Decan, and Tom Mens. 2024. Mitigating Security Risks in GitHub Actions Workflows. *Software: Practice and Experience* 54, 2 (2024), 345–362. doi:10.1002/spe.3056

[5] Omar Elazhary, Margaret-Anne Storey, Neil A Ernst, and Elise Paradis. 2021. Adept: A socio-technical theory of continuous integration. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results*. IEEE, 26–30.

[6] Omar Elazhary, Colin Werner, Ze Shi Li, Derek Lowlind, Neil A Ernst, and Margaret-Anne Storey. 2021. Uncovering the benefits and challenges of continuous integration practices. *IEEE Transactions on Software Engineering* 48, 7 (2021), 2570–2583.

[7] Brian Fitzgerald and Klaas-Jan Stol. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software* 123 (2017), 176–189.

[8] Martin Fowler and Matthew Foemmel. 2006. Continuous integration.

[9] Felix Gille, Anna Jobin, and Marcello Ienca. 2020. What we talk about when we talk about trust: theory of trust for AI in healthcare. *Intelligence-Based Medicine* 1 (2020), 100001.

[10] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*. 426–437.

[11] Jiangnan Huang and Bin Lin. 2025. Revisiting Security Practices for Github Actions Workflows. In *2025 IEEE/ACM 33rd International Conference on Program Comprehension (ICPC)*. IEEE, 73–77.

[12] Jez Humble and David Farley. 2010. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.

[13] Brittany Johnson, Christian Bird, Denae Ford, Nicole Forsgren, and Thomas Zimmermann. 2023. Make your tools sparkle with trust: The PICSE framework for trust in software tools. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 409–419.

[14] Ali Khatami, Cćdric Willekens, and Andy Zaidman. 2024. Catching smells in the act: A github actions workflow investigation. In *2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 47–58.

[15] Igibek Koishybayev, Aleksandr Nahapetyan, Raima Zachariah, Siddharth Muralee, Bradley Reaves, Alexandros Kapravelos, and Aravind Machiry. 2022. Characterizing the security of GitHub {CI} workflows. In *31st USENIX Security Symposium (USENIX Security 22)*. 2747–2763.

[16] Ravi Muralee and Others. 2023. ARGUS: Static Taint Analysis for CI/CD Vulnerabilities. In *Proceedings of the 2023 IEEE Symposium on Security and Privacy*. IEEE, 1234–1250. doi:10.1109/SP46215.2023.00075

[17] Hassan Onsori, Alexandre Decan, and Tom Mens. 2024. Quantifying Security Issues in Reusable JavaScript Actions in GitHub Workflows. In *21st International Conference on Mining Software Repositories (MSR '24)* (Lisbon, Portugal). doi:10.1145/3643991.3644899

[18] Pooya Rostami Mazrae, Tom Mens, Mehdi Golzadeh, and Alexandre Decan. 2023. On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. *Empirical Software Engineering* 28, 2 (2023), 52.

[19] Sk Golam Saroar and Maleknaz Nayebi. 2023. Developers' perception of GitHub Actions: A survey analysis. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. 121–130.

[20] Pablo Valenzuela-Toledo and Alexandre Bergel. 2022. Evolution of GitHub action workflows. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 123–127.

[21] Pablo Valenzuela-Toledo, Alexandre Bergel, Timo Kehrer, and Oscar Nierstrasz. 2023. EGAD: A moldable tool for GitHub Action analysis. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories*. IEEE, 260–264.

[22] Pablo Valenzuela-Toledo, Alexandre Bergel, Timo Kehrer, and Oscar Nierstrasz. 2023. Exploring GitHub Actions through EGAD: An Experience Report. In *Proceedings of the International Workshop on Smalltalk Technologies, Lyon, France; August 29th-31st, 2023 (CEUR Workshop Proceedings, Vol. 3627)*. CEUR-WS.org.

[23] Pablo Valenzuela-Toledo, Alexandre Bergel, Timo Kehrer, and Oscar Nierstrasz. 2024. 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, 213–223.

[24] Pablo Valenzuela-Toledo, Chuyue Wu, Sandro Hernández, Alexander Boll, Roman Machacek, Sebastiano Panichella, and Timo Kehrer. 2025. Explaining GitHub Actions Failures with Large Language Models: Challenges, Insights, and Limitations. In *2025 IEEE/ACM 33rd International Conference on Program Comprehension (ICPC)*. IEEE, 286–297.

[25] Bogdan Vasilescu, Stef Van Schuylenburg, Jules Wulms, Alexander Serebrenik, and Mark GJ Van Den Brand. 2014. Continuous integration in a social-coding world: Empirical evidence from GitHub. In *2014 IEEE international conference on software maintenance and evolution*. IEEE, 401–405.

[26] Nimmi Rashinika Weeraddana, Mahmoud Alfadel, and Shane McIntosh. 2024. Dependency-induced waste in continuous integration: An empirical study of unused dependencies in the npm ecosystem. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 2632–2655.

[27] Mohammad Yazdi. 2024. Reliability-centered design and system resilience. In *Advances in Computational Mathematics for Industrial System Reliability and Maintainability*. Springer, 79–103.

[28] Yang Zhang, Yiwen Wu, Tingting Chen, Tao Wang, Hui Liu, and Huaimin Wang. 2024. 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*. 1–13.

[29] Lianyu Zheng, Shuang Li, Xi Huang, Jiangnan Huang, Bin Lin, Jinfu Chen, and Jifeng Xuan. 2025. Why Do GitHub Actions Workflows Fail? An Empirical Study. *ACM Transactions on Software Engineering and Methodology* (2025).