

Early patch releases in npm: A retrospective study of CI and integration-test signals

Azat Davliatshin^{1,*} , Igor Kharchenko² , and Andrei Vakhitov^{2,3} 

¹*Quantori*

²*Independent Researcher*

³*Orion Innovation d.o.o., Serbia*

* Corresponding author: Azat Davliatshin, ardavlyatshin@gmail.com

OPEN ACCESS

Citation:

Azat Davliatshin, Igor Kharchenko, Andrei Vakhitov (2026). Early patch releases in npm: A retrospective study of CI and integration-test signals. *Am. Impact Rev.* e26019139.

[10.66308/air.e26019139](https://doi.org/10.66308/air.e26019139)

Received: September 30, 2026

Accepted: October 23, 2026

Published: October 25, 2026

DOI:

[10.66308/air.e26019139](https://doi.org/10.66308/air.e26019139)

ISSN: 3071-124X

Copyright:

© 2026 Azat Davliatshin. This is an open access article distributed under the terms of the Creative Commons Attribution License (CC BY 4.0).

Abstract

Early patch releases may reflect release problems or routine maintenance. This study examines associations between historical continuous integration (CI) configuration, integration-test signals, and subsequent npm patches. A fixed November 2024 sampling frame yielded a baseline of 599 releases from 216 repositories in 2022–2025. An exploratory extension added 155 releases from 70 repositories, published from January through 24 August 2026, with complete 30-day follow-up bounded by 25 September 2026. The combined dataset contains 754 releases from 217 repositories. Fourteen-day patch proportions were 32.2% and 36.8%, respectively; the difference was 4.6 percentage points (95% repository-bootstrap interval: −4.7 to 13.7). Baseline adjusted odds ratios were 1.03 (0.43–2.48) for CI configuration and 1.95 (1.21–3.13) for integration signals. Pooled estimates were 1.07 (0.44–2.57) and 2.25 (1.48–3.42). Only two 2026 releases lacked CI configuration, preventing a standalone CI comparison. Among CI-present 2026 releases, the integration-signal odds ratio was 3.34 (1.48–7.57). Models fitted exclusively to 2022–2024 releases were evaluated unchanged on both later cohorts. Adding engineering signals changed ROC-AUC from 0.678 to 0.626 in 2025 and from 0.704 to 0.749 in 2026; the 2026 difference remained uncertain. The extension retains the association between integration signals and early patching but does not establish a change in patch prevalence or consistent predictive improvement. These observational results concern release-process signals, not the causal effectiveness of testing or mandatory quality gates.

Keywords: release engineering, continuous integration, integration testing, npm, mining software repositories, software quality

1. Introduction

Preparing software for release involves decisions about automated checks, system-level testing, and how operational problems will be detected. Public repositories expose some of these decisions through configuration files, test directories, and package manifests. Package registries expose another observable event: the publication of a subsequent version. Connecting these sources creates an opportunity to examine whether practices visible before release are associated with the short-term sequence of releases received by users. It also creates a measurement problem: an easily counted patch release is not necessarily an urgent repair of a defective release.

A package can publish a patch quickly because a regression was discovered, because packaging metadata was corrected, or because the maintainers normally release small changes frequently. A capable maintenance team

may detect and distribute a correction faster than a less responsive team. Consequently, fewer early patches do not automatically imply higher reliability, and more early patches do not automatically imply poor engineering. Research on same-day npm releases already demonstrates the relevance of closely spaced publication events [1]. The present study therefore treats early patch publication as an observable release-process outcome and keeps claims about defect severity, urgency, and production incidents separate.

Continuous integration (CI) has been studied in relation to productivity, quality, and development practices [2], [3], [4]. However, the presence of a workflow is only a partial observation of the underlying process. A workflow may be optional, may cover only part of a repository, or may invoke reusable definitions that cannot be fully reconstructed from one file. Likewise, a testing dependency can exist without being executed before release. Evaluating these inexpensive signals at the release level is useful precisely because they are tempting inputs to large-scale quality assessments, despite their limited construct validity.

The research questions (RQs) are: RQ1 - How often does a stable minor or major release in the sampled cohort receive a same-line patch within 7, 14, or 30 days? RQ2 - Are historical CI configuration and integration-test signals associated with a 14-day patch after accounting for prior release cadence, package age, release type, and year? RQ3 - Do these signals improve prediction of later releases beyond a baseline based on cadence and other readily observed historical covariates? The directional expectation motivating RQ2 is a lower early-patch probability when the engineering signals are present. The analysis tests this expectation without requiring that the results support it.

The study contributes a reproducible linkage between registry publication events and historical repository evidence, a clustered observational analysis, and a temporal assessment of incremental predictive value. Its contribution is empirical and methodological; the choice of a time window alone is not presented as a novel quality metric. The cohort does not identify the first industrial deployment of each project. Throughout the paper, “minor/major release” denotes a stable version with a zero patch component rather than the first-ever release of a product.

1.1. Related work

Related work was updated through 25 September 2026. Cogo et al. studied same-day releases of popular npm packages, including their prevalence, changes, and downstream adoption [1]. Their reported median package-level proportion of regular releases interrupted by a same-day release was 22%. This provides a direct predecessor for measuring closely spaced release events. Our outcome differs by requiring a patch in the same major.minor line after a stable zero-patch anchor and by considering multiple day-based windows. More importantly, we connect the outcome to source evidence at the anchor commit, adjust for the package’s earlier release cadence, and examine predictive value on two later publication cohorts. The different sampling and outcome definitions preclude direct comparison of prevalence estimates.

Hot fixing is a broader and more heterogeneous concept than patch numbering. Hanna et al. review its terminology, techniques, and applications [5]. Their industrial study reports surveys of 24 Zühlke employees and 136 practitioners [6]. The extended mixed-method investigation adds interviews across nine project teams and documents contextual variation in repair processes and automation [7]; the overlapping surveys are not independent evidence. Hot Fixing in the Wild examines urgent repair activity using linked issues and pull requests, linguistic classification, temporal filters, and manual checks [8]. Its repair-level urgency construct differs from our package-publication outcome. These studies motivate separating a release event from the intent of a change. Neither patch numbering nor a bug-related changelog term establishes that an emergency occurred; an adjudicated urgent-fix outcome would require a separate annotation study.

Release stabilization has also been examined in project-specific settings. Rahman and Rigby investigate

stabilization in Linux and Chrome [9], while Castelluccio et al. investigate patch uplift in rapid release development pipelines [10]. Khomh et al. study the impact of rapid releases on software quality [11]. These studies establish that release timing, stabilization, and maintenance activity deserve direct investigation. Our multi-repository package cohort complements that perspective, but lacks the detailed defect and operational information available in more intensive project studies.

Studies of CI report outcomes and changes in development practice rather than treating CI as a uniform binary intervention [2], [3], [4]. The distinction matters here: a historical configuration is observable even when branch enforcement and execution history are not. Test coverage is another distinct construct. Kochhar et al. examine coverage and postrelease defects [12]; our path, dependency, and script signals do not measure coverage or fault-detection effectiveness. Accordingly, the results concern the predictive usefulness of repository-visible signals, not the efficacy of integration testing as a technique.

Version numbers also require cautious interpretation. Research on semantic versioning and package dependencies examines how versioning conventions are used across ecosystems [13], [14]. We restrict the sample to a simple numeric subset of Semantic Versioning 2.0.0 [15], excluding prereleases and pre-1.0 anchors. This supports consistent counting, but does not prove that every patch is behavior-preserving. Historical workflow datasets demonstrate that configuration evolution can be archived [16]. We retrieve the source evidence required by our independently constructed sample directly; the study does not assume that a dataset containing only CI adopters supplies a representative group of non-adopters.

2. Methods

2.1. Sampling frame and release selection

The study retains a frozen 2022–2025 baseline and adds a retrospective 2026 extension collected on 25 September 2026. Candidate membership comes from the npm-rank data release published on 27 November 2024 [17]. Its raw.json asset (release ID 187805831; asset ID 209403661) contains 10,000 returned search records and 5,247 distinct package names after deduplication. The archived generator requested forty pages of 250 npm search results with popularity weight 1 and quality and maintenance weights 0. The downloaded content matches the publisher’s SHA-256 checksum; numeric identifiers and the complete checksum are preserved in the research materials. This is a historical popularity-oriented search frame, not an exact global top-10,000 list or a probability sample of npm. Candidate names and their archived repository-link fallback are taken from this fixed artifact.

Baseline eligible anchors are versions of the form a.b.0, with integer $a \geq 1$, published from 1 January 2022 through 31 December 2025. Each requires a publication timestamp, a GitHub source reference, and a 40-character gitHead. A version-specific repository reference is preferred; if absent, the fallback is the repository URL in the archived 2024 frame. Packages are scanned in ascending SHA-256 order of the fixed seed historical-npm-rank-v1 concatenated with the package name. The first eligible package for each distinct repository is retained, and screening stops at 250 repositories. Up to four anchors per package are chosen by a second hash ordering of seed, name, and version. Selection uses eligibility and deterministic order, without conditioning on patch outcomes or successful source retrieval. The extension revisits all 250 frozen package–repository pairs, including pairs lacking usable baseline source records. It selects up to four additional anchors per pair, using the same hash seed and rules, from 1 January 2026 inclusive to 25 August 2026 exclusive. Version-specific repository references must match the frozen pair; when absent, that pair supplies the fallback. No new current-search candidates are introduced.

Baseline screening examined 584 package names, froze 714 anchors across 250 repositories, and retained 599 releases from 216 repositories after 114 source exclusions and one temporal exclusion. The extension identified

321 eligible anchors across 83 frozen pairs; the four-anchor cap retained 199. Source checks excluded 35 manifest-name mismatches and nine unavailable manifests. All 155 remaining releases, from 70 repositories, passed source-completeness and committer-time checks. Excluded anchors were not replaced. Sixty-nine repositories contribute to both periods; the pooled dataset contains 754 releases from 217 repositories. Figure 1 distinguishes the two selection paths.

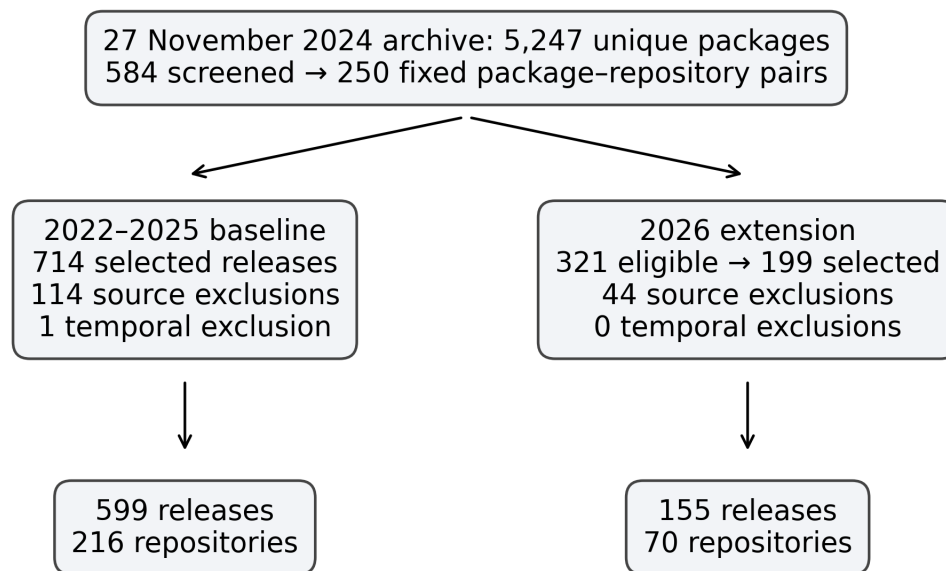


Figure 1. Selection from the fixed November 2024 sampling frame. The baseline and extension apply separate four-anchor caps. The extension covers publications from 1 January through 24 August 2026. Source exclusions concern manifest availability or package identity; temporal exclusions concern source-commit provenance.

The analysis is exploratory and was not publicly preregistered. The baseline specification and observations are retained unchanged. An extension protocol fixed the candidate pairs, hash rule, publication interval, observation endpoint, outcome windows, and comparisons before extracting 2026 outcomes. Earlier exploratory work informed the design. The same source and construct checks apply in both periods. Because at most four anchors represent four baseline years but at most four represent the partial 2026 year, period proportions are sample-dependent descriptions, not population-standardized temporal trends. Shared-repository comparisons and pooled models are additional analyses, not independent replications.

2.2. Publication events and historical source evidence

Registry metadata describes version-specific publication timestamps and repository references [18]. The gitHead field is a publisher-supplied source identifier; npm normalization can populate it from Git HEAD when absent [19]. Files are read at the recorded commit, using GitHub trees and raw source endpoints, rather than at the current default branch [20]. The cited documentation is pinned to explicit revisions for reproducibility. The package manifest at the declared repository directory, or repository root when no directory is supplied, must contain the expected package name. The source manifest version is recorded as a separate consistency check; it is not required to equal the published version in the primary analysis because release tooling may update version strings at different points in publication. An exact-version subset provides a stricter sensitivity analysis.

The temporal check compares the commit's committer timestamp with the package publication timestamp. It cannot authenticate a developer's clock or establish that registry metadata is immutable, but it rejects an observable contradiction to temporal ordering. No current branch protection setting, current star count, or later source revision is used as a pre-release predictor. CI execution outcomes are excluded because the study measures historical configuration rather than successful or mandatory execution. The analysis therefore does not depend on the retention of workflow runs, checks, or status records. Since 1 September 2026, GitHub checks, workflow runs, and statuses have been governed by Actions retention settings, capped at 90 days for public repositories [21]. Configuration at a source commit remains distinct from evidence that a check executed successfully or was mandatory.

2.3. Outcome definition

For anchor release i in repository j , let t_{ij} be its registry publication time in seconds and let P_{ij} contain stable numeric versions $a.b.p$ with $p > 0$ in the same major.minor line, published no earlier than the anchor. Registry publication-time entries are used even when the full metadata document of a version has been withdrawn. Equation (1) expresses the delay to the earliest qualifying patch in days; the minimum of an empty set is infinity. Patch numbers may be skipped, so $a.b.3$ can be the first observed patch. Publication time, rather than semantic ordering, determines the earliest subsequent event. Prereleases, parallel version lines, and patches preceding the anchor do not qualify.

$$\Delta_{ij} = \min_{v \in P_{ij}} \frac{t_v - t_{ij}}{86400}. \quad (1)$$

Equation (2) defines the binary early-patch event and the descriptive repository-level early-patch ratio. Here, N is the observation window in days, n_j is the number of analyzed anchors in repository j , and $\mathbf{1}\{\cdot\}$ is the indicator function. The primary window is 14 days; 7 and 30 days are sensitivity windows. The baseline retains its observation endpoint of 31 January 2026 at 00:00 UTC. Extension events are bounded by 25 September 2026 at 00:00 UTC, and extension anchors precede 25 August 2026 at 00:00 UTC. Both cohorts therefore have complete 30-day follow-up; later 2026 anchors are excluded rather than treated as event-free. The ratio summarizes selected anchors and is not an estimate of all historical releases in that repository. The pooled release proportion and the unweighted mean of repository ratios answer different descriptive questions, so both are reported.

$$Y_{ij}(N) = \mathbf{1}\{0 \leq \Delta_{ij} \leq N\}, \quad \text{EPR}_j(N) = \frac{1}{n_j} \sum_{i=1}^{n_j} Y_{ij}(N). \quad (2)$$

2.4. Engineering signals and covariates

CI configuration is coded as present when a recognized repository CI configuration is observed. For GitHub Actions, YAML is parsed with a non-executing loader; a push, pull-request, pull-request-target, or merge-group trigger and a recognizable build, test, lint, or check command are required. Directly invoked npm, yarn, pnpm, or bun scripts are expanded one level where the command pattern can be resolved. Conventional Travis CI, CircleCI, Azure Pipelines, GitLab CI, Jenkins, AppVeyor, and Bitbucket configuration paths are also recognized. These rules identify visible configuration, not successful execution, mandatory checks, complete workflow semantics, or protection against bypass.

The integration-test signal is deliberately broader than a claim of verified integration testing. It is present when the historical tree contains a recognized integration or end-to-end (e2e) test path, when the package manifest

declares a predefined integration/browser-testing dependency, or when a script names an integration/e2e command. The dependency list includes Playwright, Cypress, Selenium WebDriver, WebdriverIO, Testcontainers, Supertest, Puppeteer, Cucumber, Nightwatch, and Protractor. Documentation, examples, fixtures, node_modules, vendor, and website paths are excluded from path-only evidence. A separate exploratory flag records explicit integration-related commands in CI. Reusable workflows, nested scripts, provider-specific semantics, and unconventional names may remain unresolved.

An exploratory observability indicator checks direct runtime dependencies in the historical source manifest for the predefined OpenTelemetry, Sentry, prom-client, dd-trace, New Relic, and Elastic APM names. Generic logging packages and transitive dependencies are excluded. A zero means that no listed direct dependency was detected; it does not establish an absence of production observability. This indicator is reported descriptively, while the inferential models focus on CI and integration signals. In a monorepository, CI and tree-level signals can apply to sibling packages. The one-package-per-repository rule limits repeated representation but cannot remove this construct mismatch; the root-package sensitivity analysis addresses it partially.

All main covariates are available at the anchor publication: the count R_{ij} of earlier stable numeric releases in the preceding 365 days, package age A_{ij} in days since registry creation, an indicator for a major release ($b = 0$), and publication year. Release count and package age are transformed as $\ln(1 + R_{ij})$ and $\ln(1 + A_{ij})$, respectively. The tree's file count, F_{ij} , is used only in an additional size-adjusted model; it is a file-count proxy, not lines of code. Supplementary Table S1 summarizes the measurement boundaries. The data do not include production traffic, developer experience, change complexity, test coverage, defect reports, or the actual enforcement state of a release gate.

2.5. Statistical analysis and temporal evaluation

The main model is a logistic regression for the 14-day event, shown in Equation (3). Here, $p_{ij} = \Pr(Y_{ij}(14) = 1)$ is the modeled event probability, and C_{ij} and I_{ij} denote the CI and integration indicators. The control vector $\mathbf{x}_{ij}$ contains $\ln(1 + R_{ij})$, $\ln(1 + A_{ij})$, major-release status, and publication-year indicators. The intercept is β_0 ; β_C and β_I are the engineering-signal coefficients; and θ contains the control coefficients. Standard errors use a repository-cluster robust sandwich covariance estimator, allowing observations from the same repository to be dependent [22]. Reported associations are odds ratios with 95% confidence intervals, not risk ratios or causal effects. The two original primary engineering coefficients also receive Holm-adjusted p-values. Other covariates and exploratory analyses are interpreted descriptively.

$$\text{logit}(p_{ij}) = \beta_0 + \beta_C C_{ij} + \beta_I I_{ij} + \theta^\top \mathbf{x}_{ij}. \quad (3)$$

Confidence intervals for release-level event proportions use 2,000 bootstrap resamples of repositories, carrying all selected releases of a resampled repository together. The same outcome model is fitted for 7 and 30 days. Additional checks restrict the cohort to exact source-manifest version matches, restrict it to root packages, or add $\ln(1 + F_{ij})$. These models examine sensitivity to observable measurement choices; they do not eliminate residual confounding. No urgent-hotfix dictionary or manual defect label is used, and no inter-rater agreement statistic is claimed. Period differences use joint bootstrap resampling of the union of repositories, applying the same repository multiplicity to both periods to preserve cross-period dependence. A sensitivity analysis restricts both periods to their shared repositories. Pooled association models include year indicators. The 2026-only CI coefficient is not estimable because the two CI-absent releases contain no events; its integration analysis is therefore restricted to CI-present releases and omits the constant CI and year terms. This restriction changes the comparison population and is reported explicitly.

For RQ3, releases from 2022–2024 form the development set; 2025 and 2026 releases form separate temporal

evaluation cohorts. The original training data and model specification are unchanged, with no refitting or tuning on either later cohort. A baseline uses log cadence, log age, major-release status, and a numeric year trend; the expanded model adds CI and integration signals. Both use standardized inputs and L2-regularized logistic regression with fixed $C = 1$ and no holdout-driven tuning. Standardization is fitted on the development set only. The area under the receiver operating characteristic curve (ROC-AUC), average precision, and the Brier score are reported; Equation (4) defines the latter, where m is the number of holdout releases, $\hat{p}_k$ is the predicted event probability, and $y_k \in \{0, 1\}$ is the observed 14-day outcome. Lower Brier scores indicate more accurate probability forecasts. Some repositories occur in both periods, so this evaluates later releases of a partly familiar population rather than performance exclusively on unseen repositories.

$$\text{Brier} = \frac{1}{m} \sum_{k=1}^m (\hat{p}_k - y_k)^2. \quad (4)$$

Paired repository bootstrap resampling of each temporal cohort estimates uncertainty in the difference between the two fitted models. The fitted models remain fixed during this bootstrap: the intervals describe holdout-cohort uncertainty, not uncertainty from repeatedly collecting training data. Calibration is displayed using five quantile-based bins. Nine offline measurement tests cover skipped patch numbers, parallel lines, prereleases, the inclusive 14-day boundary, timestamp ordering, pre-anchor patches, anchor eligibility, the follow-up cutoff, and retained timestamps for withdrawn versions. A separate implementation recomputed the 14-day labels for all included extension observations without disagreement. Scripts, seeds, software versions, and derived data accompany the manuscript.

3. Results

3.1. Cohort characteristics and early-patch frequency

Table 1 contrasts the baseline and extension. CI configuration was present in 567 of 599 baseline releases and 153 of 155 extension releases. Integration signals were present in 128 and 62 releases, respectively; explicit integration-related CI commands occurred in 74 and 42. Prior-year release cadence was higher in the extension (median nine versus five), and integration-signal prevalence increased from 21.4% to 40.0%. These compositional differences qualify period comparisons. The runtime observability signal occurred in four baseline releases from one repository and no extension releases; it remains descriptive.

Table 1. Cohort characteristics by publication period. Percentages use releases as the denominator. IQR denotes interquartile range; 2026 covers January through 24 August.

Characteristic	2022–2025	2026
Releases / repositories	599 / 216	155 / 70
Major / minor releases	182 / 417	29 / 126
CI configuration	567 (94.7%)	153 (98.7%)
Integration-test signal	128 (21.4%)	62 (40.0%)
Explicit integration command in CI	74 (12.4%)	42 (27.1%)
Runtime observability dependency	4 (0.7%)	0 (0.0%)
Exact source-manifest version match	510 (85.1%)	136 (87.7%)
Package in repository subdirectory	105 (17.5%)	42 (27.1%)
Prior-year releases, median [IQR]	5 [1, 12]	9 [3, 28]
Package age, days, median [IQR]	2337 [1481, 3256]	3549 [2568, 4612]
Repository files, median [IQR]	73 [22, 458]	199 [52, 788]

Fourteen-day patches followed 193 of 599 baseline releases (32.2%; 95% interval, 27.5%–37.0%) and 57 of 155 extension releases (36.8%; 27.5%–46.1%). The difference was 4.6 percentage points (–4.7 to 13.7), providing no clear evidence of a period change. Table 2 also reports the seven- and thirty-day windows. Among 69 shared repositories, the corresponding rates were 33.8% (77/228) and 37.1% (56/151), a difference of 3.3 points (–5.8 to 12.4). Unweighted means of repository-specific ratios were 29.8% and 30.5% in the full baseline and extension, respectively. These differ from release-weighted proportions because repositories contribute unequal numbers of anchors. Figure 2 describes cumulative patching within each period.

Table 2. Early-patch proportions and period differences. Cells report events/releases, percentages, and 95% repository-bootstrap intervals. Difference = 2026 minus 2022–2025, in percentage points (pp).

Window	2022–2025	2026	Difference, pp
7 days	156/599; 26.0% [22.0, 30.2]	43/155; 27.7% [19.6, 36.0]	+1.7 [–6.5, 9.8]
14 days	193/599; 32.2% [27.5, 37.0]	57/155; 36.8% [27.5, 46.1]	+4.6 [–4.7, 13.7]
30 days	227/599; 37.9% [33.0, 43.0]	67/155; 43.2% [32.6, 53.6]	+5.3 [–4.3, 15.9]

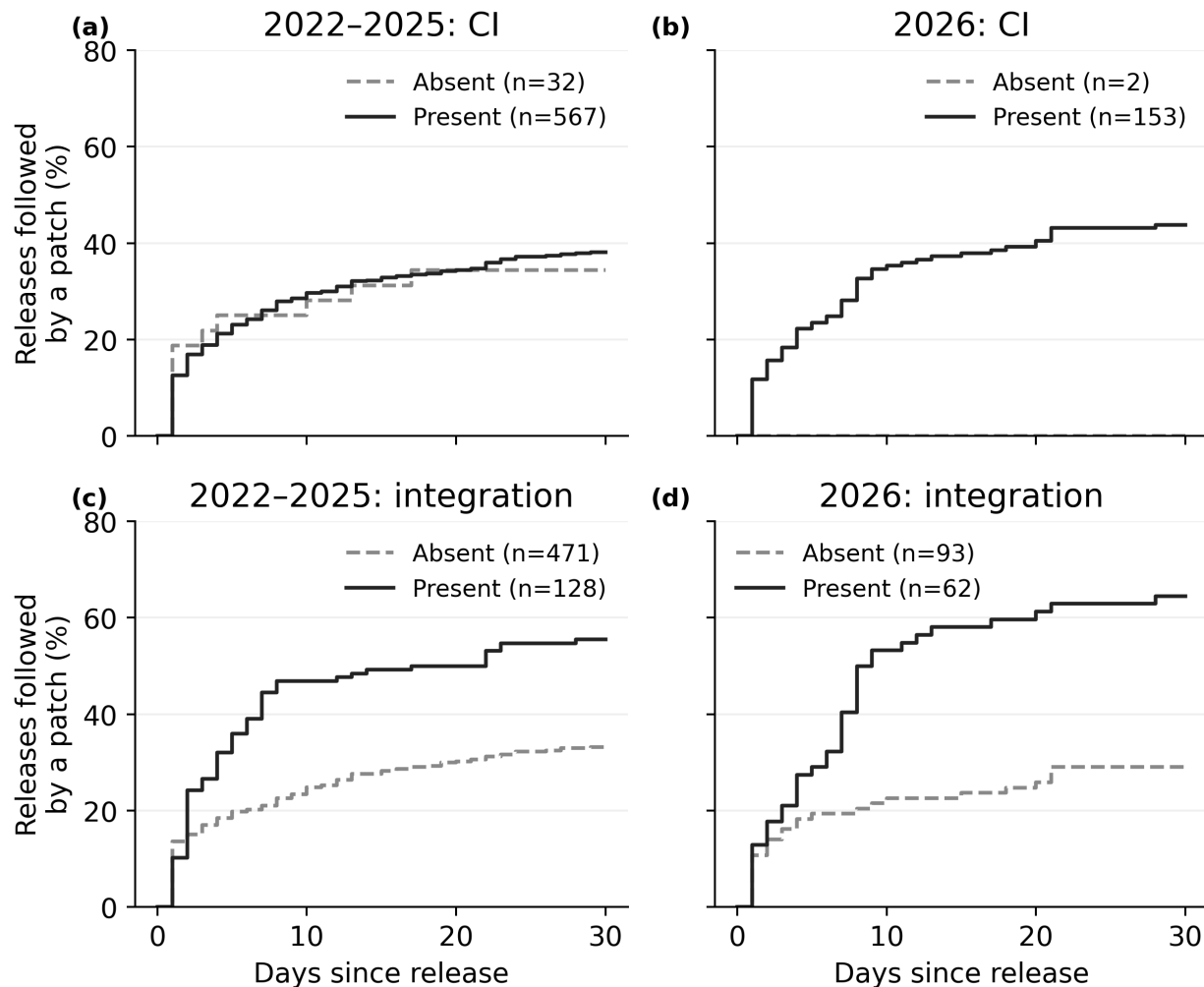


Figure 2. Cumulative patch proportions by period and engineering signal: (a, b) CI configuration; (c, d) integration signals. Group sizes are release counts. The two-release CI-absent group in 2026 does not support an inferential comparison. Curves describe sampled publication events, not causal effects.

Unadjusted 14-day proportions with and without integration signals were 49.2% and 27.6% in the baseline, and 58.1% and 22.6% in the extension. The corresponding CI contrasts were 32.3% versus 31.2% and 37.3% versus 0/2. The near-absence of CI-negative 2026 observations prevents a meaningful standalone assessment of the CI association. Raw contrasts also mix differences in release activity, package composition, and maintenance practice; they do not show that testing causes or prevents defects.

3.2. Adjusted associations

Table 3 retains the original 14-day coefficients and contrasts them with the extension and pooled analyses. Baseline adjusted odds ratios were 1.03 (0.43–2.48) for CI configuration and 1.95 (1.21–3.13) for integration signals, with Holm-adjusted p-values of 0.951 and 0.012. Pooled estimates were 1.07 (0.44–2.57) and 2.25 (1.48–3.42), respectively. Among 153 CI-present 2026 releases, the integration-signal estimate was 3.34 (1.48–7.57). This restricted estimate addresses a different population; a larger point estimate is not evidence that the association strengthened over time. CI intervals in the baseline and pooled analyses remain broad. None of these coefficients identifies the causal effectiveness of the underlying engineering practice.

Table 3. Adjusted odds ratios [95% repository-cluster intervals] for 14-day patches. The 2026 model includes only CI-present releases (n = 153); its CI coefficient is not estimable (NE). Baseline and pooled models adjust for categorical year; their year coefficients and all p-values appear in Supplementary File S1.

Predictor	2022–2025	2026, CI present	Pooled
CI configuration	1.03 [0.43, 2.48]	NE	1.07 [0.44, 2.57]
Integration signal	1.95 [1.21, 3.13]	3.34 [1.48, 7.57]	2.25 [1.48, 3.42]
Release count: $\ln(1 + R_{ij})$	1.53 [1.26, 1.87]	1.47 [1.02, 2.12]	1.54 [1.28, 1.84]
Package age: $\ln(1 + A_{ij})$	0.87 [0.75, 1.02]	0.63 [0.24, 1.66]	0.86 [0.74, 1.01]
Major vs minor	1.27 [0.82, 1.96]	1.24 [0.52, 2.97]	1.27 [0.86, 1.86]

The adjusted odds ratio per one-unit increase in $\ln(1 + R_{ij})$ was 1.53 (1.26 - 1.87). This corresponds to an odds ratio of 1.35 when one plus the prior release count doubles. Prior release cadence shows a positive adjusted association. This is a process association, not evidence that frequent releases create defects. Age, major-release status, and year effects are also estimated with uncertainty and should not be read as independent causal mechanisms. The pooled cadence odds ratio was 1.54 (1.28–1.84); the CI-present 2026 estimate was 1.47 (1.02–2.12). Figure 3 compares 14-day engineering associations across the baseline, extension, and pooled analyses. Window-specific estimates are retained in Supplementary File S1.

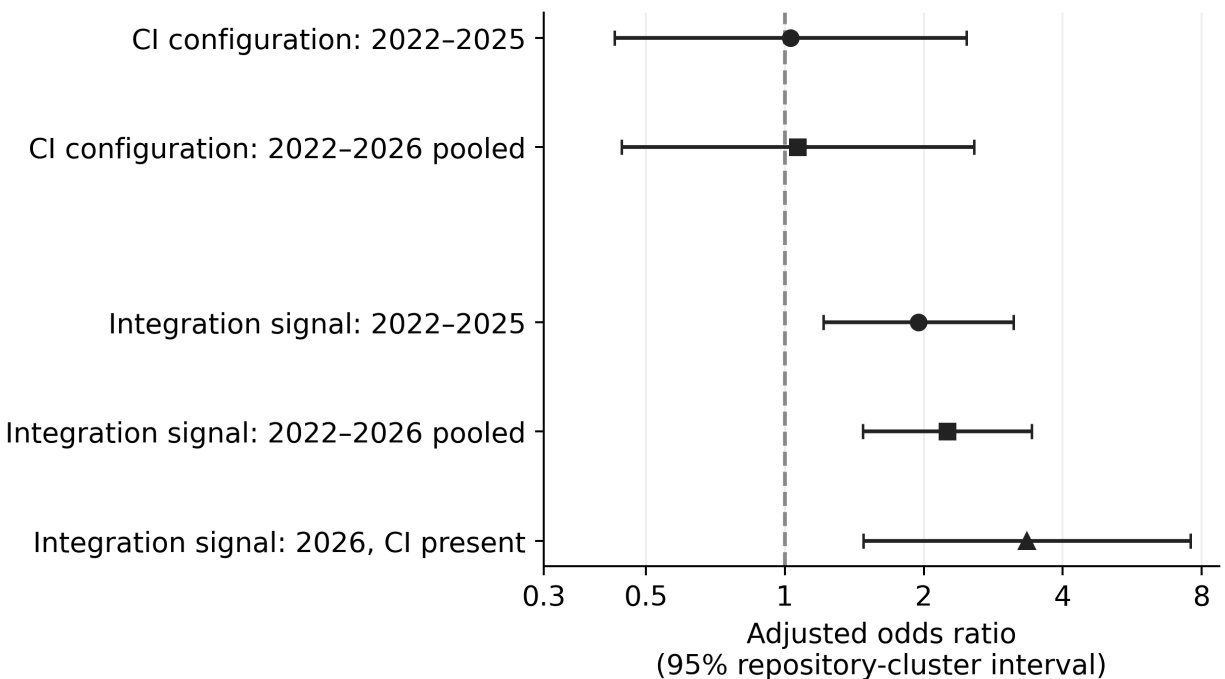


Figure 3. Adjusted 14-day associations in the baseline, pooled dataset, and CI-present 2026 subset. Bars show 95% repository-cluster intervals; the dashed line denotes OR = 1. The 2026 integration estimate conditions on CI presence and is not a direct test of a between-period coefficient difference. No standalone 2026 CI coefficient is reported.

3.3. Robustness and source availability

Integration signals retain positive associations across the declared windows, while baseline and pooled CI intervals include one. Baseline integration estimates were sensitive to exact-version linkage, root-package restriction, and file-count adjustment, with intervals including one. In the pooled dataset, the corresponding odds ratios were 1.67 (1.03–2.68), 1.69 (1.01–2.83), and 1.85 (1.21–2.83). In the CI-present extension subset, estimates were 2.61 (1.12–6.11), 3.47 (1.25–9.59), and 3.14 (1.37–7.18). Supplementary File S1 reports the full estimates. These overlapping analyses examine measured selection and size differences; they do not remove residual confounding or supply independent confirmation.

Source availability is a material selection issue. The 115 frozen-sample anchors omitted from the final analysis had a 14-day patch proportion of 43.5%, compared with 32.2% among retained anchors. Their median prior-year release count was 9, compared with 5 for retained anchors. These descriptive comparisons do not establish that missingness is random. Missing records were not coded as having no CI or no integration testing; their observable registry outcomes are retained in the missingness-comparison table. In the extension, the 44 source-excluded anchors had 20 events (45.5%) and a median prior-year release count of 16.5, compared with 57 events (36.8%) and a median of nine among included anchors. Source availability therefore remains a material selection limitation in both periods.

3.4. Temporal predictive value

The unchanged development set contained 492 releases from 203 repositories. The 2025 evaluation retained 107 releases from 80 repositories, 67 shared with development; its prevalence was 29.9%. The 2026 evaluation contained 155 releases from 70 repositories, 63 shared with development, with prevalence 36.8%. Adding engineering indicators reduced 2025 ROC-AUC by 0.0521 (95% paired interval, –0.1042 to –0.0146). On 2026 releases, ROC-AUC increased from 0.7044 to 0.7490, a difference of 0.0446 (–0.0087 to 0.1042). Table 4 reports all metrics using the same two fitted models.

Table 4. Temporal performance of unchanged models trained on 2022–2024 releases. Difference = expanded minus baseline; lower Brier scores are better. Paired repository-bootstrap intervals condition on the fitted models.

Evaluation / metric	Baseline	Expanded	Difference [95% interval]
2025 / ROC-AUC	0.6779	0.6258	–0.0521 [–0.1042, –0.0146]
2025 / Average precision	0.5430	0.5040	–0.0390 [–0.1089, +0.0114]
2025 / Brier score	0.1910	0.1991	+0.0082 [–0.0040, +0.0220]
2026 / ROC-AUC	0.7044	0.7490	+0.0446 [–0.0087, +0.1042]
2026 / Average precision	0.6003	0.6374	+0.0371 [–0.0295, +0.1264]
2026 / Brier score	0.2065	0.1930	–0.0135 [–0.0271, +0.0005]

The direction of incremental ROC-AUC differed between evaluation cohorts. In 2026, expanded-model point estimates also improved average precision and the Brier score, but all three difference intervals included zero. The extension therefore does not establish a consistent predictive benefit across periods. Calibration in Figure 4 is descriptive: each panel uses five quantile bins, with bin counts retained in the replication data. The models are research comparisons rather than validated release-blocking tools.

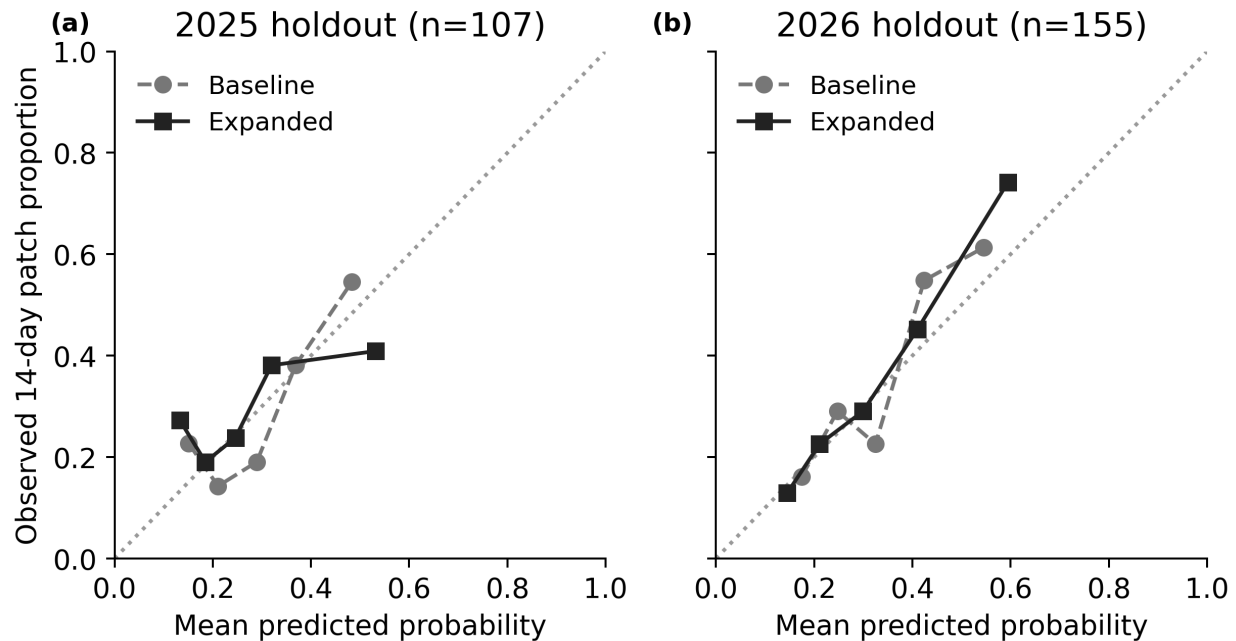


Figure 4. Calibration in (a) the 2025 and (b) the 2026 evaluation cohorts for models trained exclusively on 2022–2024 releases. Each curve uses five quantile bins; the diagonal denotes ideal calibration. Bin sizes are limited, and no model was recalibrated on these outcomes.

4. Discussion

4.1. Implications for release preparation

The results distinguish engineering practices from the inexpensive signals used to observe them. CI configuration is common in this selected cohort, limiting contrast between presence and absence. Integration signals combine tools with different purposes and do not establish how thoroughly components were exercised. CI configuration did not show a clear adjusted association in the baseline or pooled analyses; a 2026-only CI comparison was not estimable. Integration signals were associated with higher early-patch odds in both periods, including the CI-present extension subset. The operationalization therefore cannot certify production readiness or isolate the effect of a testing technique. Interpreting the estimates requires the event definition, comparison-group size, and uncertainty intervals together.

Release cadence provides essential context for interpreting an early-patch ratio. A team that publishes small increments frequently has more opportunities to publish a patch within a fixed window. Another team may retain a known fix until a scheduled release outside that window. Both maintenance behavior and detection speed influence the outcome. Prior release cadence shows a positive adjusted association. A release-quality assessment should combine publication events with evidence about the reason for an update, regression severity, and user impact. This study observes only the publication part of that chain.

For repository-based assessments, the findings favor preserving evidence levels: a configuration exists; it defines relevant test commands; the commands execute on the relevant source; and their success is required before a release proceeds. Each is a stronger claim than the preceding one and requires additional observations. Our data substantiate parts of the first two levels. Historical branch enforcement, execution success, bypass permissions, and actual production readiness remain unobserved. Describing the current analysis as a test of mandatory CI gates would overstate what was measured.

4.2. Measurement and reproducibility

The analysis separates prevalence, adjusted association, and prospective-style temporal prediction because they answer different questions. A sizeable event proportion establishes that early patches are common in the sampled process. An association model examines measured conditional relationships. A holdout comparison tests whether additional features improve forecasts for later releases. Statistical association and engineering plausibility do not, by themselves, establish incremental predictive value. Conversely, a small predictive gain in one cohort does not establish that the underlying practice lacks value for maintenance, reliability, or developer feedback. The two later cohorts illustrate why temporal evaluations should be repeated: the incremental ROC-AUC changed direction, and its 2026 interval included zero. The newer industrial evidence likewise emphasizes contextual variation in hot-fix practice [7], rather than a uniform relationship between automation and post-release repair.

Temporal provenance is a substantive measurement requirement. Reading workflow files or branch protection from the current default branch could assign practices to releases that preceded their adoption. The archived sampling frame and source-commit checks address different risks: the former fixes candidate membership, while the latter constrains the source revision used for each release. A gitHead link and matching manifest name improve traceability but do not provide the assurance of a verified build attestation or trusted archival timestamp. The exact-version sensitivity analysis makes part of this remaining uncertainty visible.

The observability indicator was present in four baseline releases across one repository and absent from the extension. This package-centered sample is a limited basis for generalizing about runtime monitoring in deployed applications. Instrumentation can be supplied by a host application, an agent, or deployment infrastructure outside the source manifest. Its descriptive count is therefore not an estimate of production observability coverage. An application-oriented study would need a population and evidence source appropriate to that construct.

4.3. Limitations and threats to validity

Construct validity. An early patch can be planned maintenance, a packaging repair, a security update, or an urgent regression fix. We do not have adjudicated labels distinguishing them. Stable numeric version parsing excludes some real release schemes and does not establish compliance with semantic-versioning promises. CI and integration indicators are heuristic: scripts can be nested, test names can be unconventional, and reusable workflows may be only partly visible. Tool dependencies can be unused or serve unrelated purposes. No independently coded human validation set or inter-rater statistic is available, so classifier precision and recall remain unknown.

Internal validity. The design is observational and cannot identify the causal effect of adopting CI or integration testing. Team capacity, change size, package complexity, adoption, testing maturity, and maintenance policy can affect both the indicators and patch behavior. Adjustment for cadence, age, type, and year is limited. Clustering addresses within-repository dependence in uncertainty estimation; it does not remove confounding. Current registry metadata and historical git objects can also be modified or become unavailable. The committer-time check reduces one observable temporal problem but is not a trusted timestamping service.

External validity. The fixed 27 November 2024 popularity-oriented frame emphasizes established, discoverable packages and excludes packages absent from that archive. Its duplicate search records do not justify describing the deduplicated set as an exact global ranking. Requiring gitHead, accessible source, and a matching manifest adds further selection. Hash ordering makes selection reproducible, but package ordering before repository deduplication does not give every repository equal inclusion probability: repositories publishing many eligible packages can be encountered earlier. The 2024 membership criterion can also favor projects surviving from

the earlier anchor years. Results should not be generalized to all npm packages, other ecosystems, enterprise systems, or first industrial deployments. The 2026 extension further selects packages that continued to issue eligible zero-patch releases. Separate four-anchor caps over unequal calendar spans prevent interpreting the raw period difference as an ecosystem-wide trend; the shared-repository analysis only partly addresses composition.

Statistical conclusion validity. CI-absent releases are sparse in the baseline and virtually absent in the extension. Quasi-complete separation precludes a conventional standalone 2026 CI estimate; restricting its integration analysis to CI-present releases changes the estimand. Failure to reject a null hypothesis is not equivalence. Additional models and windows are overlapping exploratory analyses. Two partial-overlap temporal cohorts do not establish durable predictive generalization, and their bootstrap intervals exclude training uncertainty. At most four anchors per period cannot characterize every phase of project evolution. No formal interaction test establishes that an engineering association changed between periods.

Reproducibility and reporting. Archived candidate names, publication timestamps, source identifiers, extraction scripts, derived observations, model tables, and figure outputs are retained. The replication materials include the frame's release and asset identifiers and checksum. They exclude credentials and package-maintainer email records. Live source retrieval can differ if registry records or repositories become unavailable; rerunning analysis of the frozen observations is the reproducible target. The exploratory status and measurement choices are explicit, and the primary 14-day outcome is retained regardless of the direction of the estimated associations. The September amendment, original records, new records, and unchanged-model evaluations are stored separately so that the baseline findings remain reproducible.

5. Conclusions and future work

The September 2026 extension adds 155 releases to a frozen baseline of 599, producing 754 observations from 217 repositories. Fourteen-day patch proportions were 32.2% in 2022–2025 and 36.8% in 2026, with an uncertain between-period difference. The positive integration-signal association persisted in pooled and CI-present extension analyses; CI configuration had no clear baseline or pooled association and was not separately estimable in 2026. Adding engineering signals worsened temporal ROC-AUC in 2025 and improved its point estimate in 2026, with the latter difference remaining uncertain. Early-patch frequency consequently requires release-process context, and configuration presence remains a limited proxy for preparation rather than evidence of production readiness.

Future work should validate urgent and regression-related patches using independently coded release narratives, strengthen source linkage with verified provenance, and collect enforcement and execution evidence prospectively. Broader sampling should include projects outside the historical popularity-oriented frame and applications whose runtime observability can be meaningfully observed. A larger non-CI comparison group and repeated temporal evaluations would improve precision. Such extensions should preserve the distinction between predictive association, a release-event definition, and causal claims about engineering practices.

Acknowledgments

AI assistance

Numerical analyses and figures were generated using the accompanying Python scripts applied to public registry and repository records. Reference DOIs were verified against bibliographic metadata. AI assistance does not constitute independent human validation of the engineering labels. Responsibility for the accuracy, interpretation, and final approval of the manuscript remains with the author.

Data availability

Supplementary File S2 retains the fixed sampling frame, baseline records, September extension protocol, new release-level observations, source-evidence audits, committer-time checks, analysis scripts, model outputs, and figures. Baseline event histories end on 31 January 2026 at 00:00 UTC; extension histories end on 25 September 2026 at 00:00 UTC. Extension metadata and historical source files were retrieved on 25 September 2026. Source revisions are identified by commit hashes, and the analyses can be reproduced offline from derived records. A permanent public repository link and archival DOI should be supplied when the author deposits the reviewed materials; no deposit is claimed here.

Supplementary materials

Supplementary File S1 contains the measurement dictionary, window-specific association models, sensitivity analyses, shared-repository comparison, and source-availability summaries. Supplementary File S2 contains frozen baseline and extension data, code, and verification records. Four high-resolution PNG figures accompany the manuscript.

Declarations

Ethics approval and consent. This computational study analyses public software-release records and repository artefacts. It did not involve recruitment of human participants or experiments involving animals; ethics approval and informed consent were therefore not applicable to the study design.

References

- [1] F. R. Cogo, G. A. Oliva, C. P. Bezemer, and A. E. Hassan, “An empirical study of same-day releases of popular packages in the npm ecosystem,” *Empirical Software Engineering*, vol. 26, no. 5, Art. no. 89, 2021. <https://doi.org/10.1007/s10664-021-09980-6>
- [2] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, “Quality and productivity outcomes relating to continuous integration in GitHub,” *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 805 - 816, 2015. <https://doi.org/10.1145/2786805.2786850>
- [3] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, “Usage, costs, and benefits of continuous integration in open-source projects,” *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 426 - 437, 2016. <https://doi.org/10.1145/2970276.2970358>
- [4] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu, “The impact of continuous integration on other software development practices: A large-scale empirical study,” *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 60 - 71, 2017. <https://doi.org/10.1109/ASE.2017.8115619>
- [5] C. Hanna, D. Clark, F. Sarro, and J. Petke, “Hot Fixing Software: A Comprehensive Review of Terminology, Techniques, and Applications,” *ACM Transactions on Software Engineering and Methodology*, early access, Art. no. 3786330, 2025. <https://doi.org/10.1145/3786330>
- [6] C. Hanna, D. Elliman, W. Emmerich, F. Sarro, and J. Petke, “Behind the Hot Fix: Demystifying Hot Fixing Industrial Practices at Zühlke and Beyond,” *Proceedings of the 33rd ACM International*

- Conference on the Foundations of Software Engineering, FSE Companion 2025, pp. 411 - 421, 2025. <https://doi.org/10.1145/3696630.3728564>
- [7] C. Hanna, D. Elliman, W. Emmerich, F. Sarro, and J. Petke, “A Mixed-Method Study of Hot Fixing in Industry: Practices, Bottlenecks, and Opportunities,” *ACM Transactions on Software Engineering and Methodology*, early access, 2026. <https://doi.org/10.1145/3816247>
 - [8] C. Hanna, K. Even-Mendoza, W. B. Langdon, M. Zamorano López, J. Petke, and F. Sarro, “Hot Fixing in the Wild,” arXiv:2604.26892, preprint, 2026. <https://doi.org/10.48550/arXiv.2604.26892>
 - [9] M. T. Rahman and P. C. Rigby, “Release Stabilization on Linux and Chrome,” *IEEE Software*, vol. 32, no. 2, pp. 81 - 88, 2015. <https://doi.org/10.1109/MS.2015.31>
 - [10] M. Castelluccio, L. An, and F. Khomh, “An empirical study of patch uplift in rapid release development pipelines,” *Empirical Software Engineering*, vol. 24, no. 5, pp. 3008 - 3044, 2019. <https://doi.org/10.1007/s10664-018-9665-y>
 - [11] F. Khomh, B. Adams, T. Dhaliwal, and Y. Zou, “Understanding the impact of rapid releases on software quality: The case of Firefox,” *Empirical Software Engineering*, vol. 20, no. 2, pp. 336 - 373, 2015. <https://doi.org/10.1007/s10664-014-9308-x>
 - [12] P. S. Kochhar, D. Lo, J. Lawall, and N. Nagappan, “Code Coverage and Postrelease Defects: A Large-Scale Study on Open Source Projects,” *IEEE Transactions on Reliability*, vol. 66, no. 4, pp. 1213 - 1228, 2017. <https://doi.org/10.1109/TR.2017.2727062>
 - [13] A. Decan and T. Mens, “What Do Package Dependencies Tell Us About Semantic Versioning?” *IEEE Transactions on Software Engineering*, vol. 47, no. 6, pp. 1226 - 1240, 2021. <https://doi.org/10.1109/TSE.2019.2918315>
 - [14] D. Pinckney, F. Cassano, A. Guha, and J. Bell, “A Large Scale Analysis of Semantic Versioning in NPM,” 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp. 485 - 497, 2023. <https://doi.org/10.1109/MSR59073.2023.00073>
 - [15] T. Preston-Werner, “Semantic Versioning 2.0.0,” specification, revision 7c834b3f3a49, Jun. 18, 2013. [Online]. Available: <https://github.com/semver/semver/blob/7c834b3f3a4940d77ab593bc32583004d6a426a9/semver.md> Accessed: Oct. 25, 2026.
 - [16] G. Cardoen, T. Mens, and A. Decan, “A dataset of GitHub Actions workflow histories,” *Proceedings of the 21st International Conference on Mining Software Repositories*, pp. 677 - 681, 2024. <https://doi.org/10.1145/3643991.3644867>
 - [17] npm-rank contributors, “NPM Ranking,” data release 187805831, raw.json asset 209403661, Nov. 27, 2024. [Online]. Available: <https://api.github.com/repos/tristan-f-r/npm-rank/releases/assets/209403661> Accessed: Oct. 25, 2026.
 - [18] npm, “Package metadata,” revision ae49abf1bac0, May 7, 2024. [Online]. Available: <https://github.com/npm/registry/blob/ae49abf1bac0ec1a3f3f1fceealcca6fe2dc00e1/docs/responses/package-metadata.md> Accessed: Oct. 25, 2026.
 - [19] npm, “Package normalization: gitHead population,” revision 57952b808317, Nov. 18, 2025. [Online]. Available: <https://github.com/npm/package-json/blob/57952b808317d8495750f195158c6bb7f4704a63/lib/normalize.js#L492-L539> Accessed: Oct. 25, 2026.

- [20] GitHub, “GitHub v3 REST API: OpenAPI description,” revision 61cae031ab56, Apr. 14, 2026. [Online]. Available: <https://github.com/github/rest-api-description/blob/61cae031ab56983c1172900cb2858c0d8998976f/descriptions/api.github.com/api.github.com.json> Accessed: Oct. 25, 2026.
- [21] GitHub, “Actions retention now covers checks, runs, and statuses,” GitHub Changelog, Oct. 1, 2026. [Online]. Available: <https://github.blog/changelog/2026-10-01-actions-retention-now-covers-checks-runs-and-statuses/> Accessed: Oct. 25, 2026.
- [22] A. C. Cameron and D. L. Miller, “A Practitioner’s Guide to Cluster-Robust Inference,” *Journal of Human Resources*, vol. 50, no. 2, pp. 317 - 372, 2015. <https://doi.org/10.3368/jhr.50.2.317>