

Specification Artifacts in Open-Source Pull Request Workflows Do Not Reduce Defects: An Empirical Test of Spec-Driven Development Claims

Brenn Hill  

Independent Researcher, Germany

Abstract

Context. Spec-driven development (SDD) tools claim that writing specifications before implementation reduces defects, prevents rework, and improves code quality. No vendor has published empirical evidence for these claims.

Method. We test five hypotheses derived from SDD vendor claims against 88,052 pull requests across 119 open-source repositories. What we measure is specification *artifacts* – overwhelmingly references to tracking issues and tickets – rather than SDD tool output, which no repository in the sample commits. We score 25,209 artifacts on the same quality dimensions SDD tools prescribe, measure rework from subsequent pull request activity, trace defects via the SZZ algorithm, and compare each developer’s spec’d PRs to their own unspec’d PRs. Twelve robustness checks include propensity score matching, complexity stratification, and AI/human subgroup analysis.

Results. None of the five hypotheses are supported. After propensity score matching on just-in-time risk profiles, spec’d and unspec’d PRs are indistinguishable in defect introduction (-0.6 percentage points (pp), $p = 0.133$). Rework shows no protective effect either ($+1.2$ pp, $p = 0.001$ within-author; $+0.5$ pp, $p = 0.146$ matched). Specification quality predicts neither defects ($p = 0.164$) nor rework ($p = 0.860$), and specifications do not constrain the scope of AI-tagged changes ($p = 0.997$). The positive raw association between specification artifacts and defects is explained by task complexity: developers spec their hardest work. One subgroup runs the other way – in repositories with no detected AI use, specifications are associated with less rework (-3.3 pp, $p = 0.014$) – but it is one of six subgroup tests and does not survive correction for multiple comparisons.

Conclusion. Specification artifacts proxy for task complexity, not quality improvement. SDD tool adoption itself remains untested: no repository in the sample commits such output.

2012 ACM Subject Classification Software and its engineering → Software creation and management

Keywords and phrases Spec-driven development, specification artifacts, null result, defect prediction, propensity score matching, SZZ, AI-assisted development, within-author fixed effects

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.84

Category Software Engineering in Practice Track Paper

Related Version *Full Version:* <https://zenodo.org/records/19432099>

Supplementary Material *Dataset (Dataset and Replication Package):* <https://doi.org/10.5281/zenodo.19415187>

1 Introduction

The rise of AI-assisted software development has produced a new category of developer tooling: specification-driven development (SDD) frameworks. These tools – including GitHub’s Spec Kit [8] and Amazon’s Kiro [6] – claim that writing formal specifications before implementation leads to fewer defects, less rework, and higher-quality code, particularly when AI agents perform the implementation.



© Brenn Hill;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).
Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;
Article No. 84; pp. 84:1–84:20



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Böckeler [4] defines spec-driven development as “writing a “spec” before writing code with AI” where “[t]he spec becomes the source of truth for the human and the AI,” and examines three tools pursuing the approach. The GitHub Copilot Academy frames it more strongly: “Specifications don’t serve code – code serves specifications” [1]. More broadly, SDD is positioned as an antidote to “vibe coding” – iteratively prompting AI without formal requirements – which practitioners associate with scope creep, rework, and defects.

The core premise is: if developers articulate what software should do before writing it, AI systems will produce more reliable output. Vendors state this as a set of specific, testable quality claims – fewer defects, less rework, higher accuracy, fewer costly rewrites [1, 23, 8] – which we quote verbatim and map to hypotheses in Section 3. They rest on a plausible causal chain: clearer requirements → less ambiguity → fewer defects. Yet vendors cite no empirical evidence for them in their marketing materials, documentation, or blog posts, and no study has tested SDD’s quality claims on production-scale data prior to this paper.

This paper provides the first large-scale empirical test. We analyze 88,052 merged pull requests across 119 open-source repositories, after excluding 12,195 bot-authored PRs. Defects are traced to their introducing commits using the SZZ algorithm (named for Śliwerski, Zimmermann, and Zeller), which works backward from bug-fixing commits through version control history to identify which earlier commits introduced each bug. We compare each developer’s spec’d PRs to their own unspec’d PRs to control for developer-level confounding, and derive five hypotheses directly from vendor claims. None of the five survive: once task complexity is accounted for, specification artifacts are unrelated to defects, rework, or the scope of AI-tagged changes.

2 Related Work

2.1 Requirements Engineering and Defect Prevention

The claim that upfront specification prevents defects has deep roots. Boehm’s cost-of-change curve [5] argued that requirements defects are exponentially cheaper to fix early, but subsequent work challenged its universality: Shull et al. [21] found the cost multiplier depends on project context, and agile methods showed iterative refinement can outperform upfront specification [3]. Direct tests of whether requirements quality predicts defects have found weak or context-dependent effects: Mund et al. [15] found only weak evidence across two studies, and Femmer et al. [10] found that the relationship between requirements smells and downstream defects was context-dependent. Our study confirms this pattern at larger scale.

2.2 SDD Literature

The academic literature on SDD perpetuates the pattern of unsubstantiated claims. Piskala [18] claims “controlled studies showing error reductions of up to 50%,” citing a Red Hat Developer blog post [16] and an InfoQ article [11]; neither contains any controlled study, experiment, or quantitative data, so the figure traces through a citation chain resting on practitioner opinion. Guo et al. [12] claim SDD “guarantees software quality” without empirical validation. Marri [14] reports a 73% security defect reduction from constitutional specs, but on a single project with the same developer implementing both conditions sequentially, acknowledging that “small sample size limits statistical power.” Roy [19] proposes iterative spec-code co-evolution but reports no independently validated outcomes. Zietsman [25] runs small planted-bug experiments, is honest that they are “directional, not statistically significant,” and contributes the useful observation that AI reviewers without specs suffer circular reasoning – but does not test defect reduction at scale.

Practitioner evaluations have been more skeptical. Eberhardt found Spec Kit roughly ten times slower than iterative development, producing thousands of lines of specification markdown that still resulted in buggy code [9]. Zaninotto argued that “SDD shines when starting a new project from scratch, but as the application grows, the specs miss the point more often and slow development” [24].

3 Hypotheses

We identified five empirical claims made by SDD tool vendors in their public documentation, blog posts, and marketing materials. Each claim is stated verbatim and mapped to a testable hypothesis.

SDD as vendors describe it is a chain with two links: a structured specification is authored before implementation, and an AI agent then implements it. Our data supports a test of the first link only. We observe whether a specification artifact exists and what it contains, but we cannot observe whether a tool generated it or whether an agent consumed it – and no repository in our sample commits SDD tool output (Section 6.3). The hypotheses below are therefore stated as vendors state them, because testing their claims as made is the point, but what they are tested *against* is the presence and quality of specification artifacts in ordinary pull request workflows. Section 7 returns to the distance between the two.

Claim 1: Specifications reduce defects

GitHub’s Spec Kit blog post promises “less guesswork, fewer surprises, and higher-quality code” [8]. Kiro claims specifications lead to “fewer iterations and higher accuracy” [23].

H1. Pull requests with specification artifacts have lower defect-introduction rates than pull requests without them, controlling for author and PR size.

Claim 2: Specifications reduce rework

The GitHub Copilot Academy lists “reduced rework” among SDD’s benefits [1]. Kiro claims to “prevent costly rewrites” [23]. GitHub’s blog states that SDD enables “iterative development without expensive rewrites” [8].

H2. Pull requests with specification artifacts have lower rework rates than pull requests without them, controlling for author and PR size.

Claim 3: Clearer specifications produce more reliable code

GitHub’s Spec Kit blog states that “providing a clear specification up front, along with a technical plan and focused tasks, gives the coding agent more clarity, improving its overall efficacy” [8]. Kiro claims specifications enable “fewer iterations and higher accuracy” [23]. This implies a dose-response relationship: higher specification quality should predict fewer defects.

H3. Among pull requests with specifications, higher specification quality predicts lower defect-introduction rates, controlling for author and PR size.

H4. Among pull requests with specifications, higher specification quality predicts lower rework rates, controlling for author and PR size.

Claim 4: Specifications constrain AI-generated code scope

GitHub’s Spec Kit blog describes tasks that give “the coding agent a way to validate its work and stay on track” [8]. Kiro claims specs provide “a North Star to guide the work of the agent, allowing it to take on larger tasks without getting lost” [6].

H5. Specifications reduce code churn (additions + deletions) for AI-tagged pull requests more than for human-authored pull requests, controlling for author.

4 Methods**4.1 Dataset**

We collected pull request data from 119 open-source repositories spanning 17 programming languages, between March 28 and April 3, 2026, with a 365-day lookback window per repository. The resulting dataset spans April 2025 through early April 2026 (1st–99th percentile: April 8, 2025 to April 1, 2026), totaling 100,247 PRs. We excluded 12,195 bot-authored PRs (dependabot, renovate, and other automated accounts identified by username pattern and bot flags), leaving 88,052 PRs from human and AI-assisted authors.

For each pull request, we extracted: author, merge date, additions, deletions, files changed, linked issues, PR description, review comments, and co-author tags (used to identify AI-assisted contributions).

4.1.1 Sampling Frame and Inclusion Criteria

Repositories were selected purposively rather than randomly, and the sample was stratified on the constructs of interest rather than drawn from an enumerated population. The first wave of 43 repositories was stratified on specification practice, using issue-linking rate as the observable proxy: near-universal linking (8 repositories), 60–90% (12), 30–50% (6), and low linking (3), plus a stratum of 14 repositories with visible AI adoption. A second wave of 77 repositories, each verified on March 26, 2026 to have at least 200 merged pull requests in the preceding year, extended the same design with 33 repositories of expected strong process enforcement, 24 of weak or unknown enforcement, and 20 with heavy AI adoption; this wave also broadened language and domain coverage so that no single ecosystem would dominate. One further repository returned no merged pull requests in the observation window and is absent from the analysis, leaving 119.

Two features of this process require disclosure. First, the sample is deliberately *enriched* for strong specification practice: the strong-enforcement and near-universal-linking strata were selected precisely because they were expected to exhibit the benefit SDD vendors claim. Enrichment of this kind biases toward detecting an effect, not toward the null we report. Second, the sample size was not fixed in advance. Repositories were added while no effect was evident, and collection stopped at 119 once the null appeared stable across the checks in Section 5.6. Continuing to add data in search of an effect and failing to find one is conservative with respect to our central claim, but it does mean that sample composition is not independent of the analyses, and that nominal *p*-values should not be read as exactly calibrated. Optional stopping of this kind inflates false positives rather than false negatives, so it bears most directly on the one positive subgroup result (Section 5.6.6), which we already treat as suggestive rather than confirmatory.

The sample spans 17 programming languages, with Python (31 repos, 26%), TypeScript (26, 22%), Go (17, 14%), and Rust (13, 11%) predominating. The median repository has a 26.2% specification rate (IQR 16.0–41.1%). AI tagging is present in most repositories: the

■ **Table 1** Study dataset overview.

Metric	Value	Notes
Repositories	119	purposive sample, 17 languages
PRs (post-bot exclusion)	88,052	12,195 bots removed
Median PRs per repo	517	range 9–4,803
Median spec rate (per repo)	26.2%	IQR 16.0–41.1%
Median AI tag rate (per repo)	1.2%	IQR 0.4–8.1%
SZZ-covered repos	103	16 had unreachable merge SHAs
SZZ bug-introducing PRs	9,754	12.5% of SZZ-covered PRs
Quality-scored PRs	25,209	28.6% of total; 17,094 enriched with issue body

■ **Table 2** Decomposition of the `spec'd` treatment by the criterion that fired (88,052 non-bot PRs, of which 24,298 are spec'd).

Detection criterion	PRs	% of spec'd
GitHub issue #N with linking keyword	11,348	46.7%
GitHub issue URL	7,252	29.8%
Project-key ticket ID (PROJ-123)	5,112	21.0%
External document URL (Docs, Linear, Confluence, Notion, Jira)	561	2.3%
Filled requirements section in PR body	24	0.1%
Total	24,298	100%

median AI tag rate is 1.2% (IQR 0.4–8.1%). The 5,989 AI-tagged PRs support the primary AI analyses, though a larger AI-tagged sample would be preferable as agentic workflows spread.

4.2 Variables

4.2.1 Specification Presence (`spec'd`)

A pull request is classified as “specified” (hereafter “spec’d”) if its description or title references an upfront specification artifact. Classification is programmatic. We accept a GitHub issue reference (#123) only when it appears alongside a linking keyword (`fixes`, `closes`, `resolves`, `references`, `related to`) and outside blockquoted template examples; project-key ticket identifiers (PROJ-123) after filtering common false positives such as `RFC-`, `CVE-`, and `GPT-`; URLs to issue trackers and document hosts (GitHub issues, Jira, Confluence, Linear, Notion, Shortcut, Google Docs), excluding issue-*creation* template links; and filled requirements or specification sections in the PR body containing at least 20 words of non-placeholder content. 24,298 of 88,052 PRs (27.6%) are classified as specified.

Table 2 decomposes the treatment by which criterion fired, and the result governs how everything downstream should be read: 97.5% of spec’d PRs are identified by a reference to a tracking issue or ticket, while structured requirements text written directly into the PR body accounts for 24 PRs. Whatever else this measure captures, in practice it measures *issue- and ticket-reference presence*. We use the term “specification artifact” throughout in that concrete sense.

This operationalization is deliberately generous, capturing any form of upfront specification artifact (Section 7). Specification *presence* and *quality* are tested separately: H1/H2 test whether having any spec predicts fewer defects or less rework; H3/H4 test whether, among spec'd PRs, higher quality predicts better outcomes.

4.2.2 Specification Quality (q_overall)

For the subset of specified PRs with substantial descriptions (body length > 50 characters), we scored specification quality by prompting Claude Haiku (Anthropic) with the PR title and body and a structured rubric. The LLM scored each PR across seven dimensions: outcome clarity, error states, scope boundaries, acceptance criteria, data contracts, dependency context, and behavioral specificity. Each dimension is scored 0–100, and the overall quality score is the mean across dimensions.

The dimensions deliberately mirror what SDD tools themselves prescribe, so that we measure quality on the criteria vendors say matter. Spec Kit's template asks for "core requirements" (our outcome clarity), "acceptance criteria," "edge cases" (error states), "data models" and "API contracts" (data contracts), and "user stories" (behavioral specificity) [8]; Kiro asks for "clear requirements," "acceptance criteria," "system architecture," and "technology choices" [23]. Six of our seven have direct equivalents in Spec Kit's structure, and all seven are covered across both tools.

We validated the scoring against independent human ratings on a stratified random sample of 38 PRs, blind to LLM scores. Overall agreement was moderate (Spearman $\rho = 0.42$, $p = 0.01$; Pearson $r = 0.45$, $p < 0.01$), but uneven across dimensions: four agree significantly (behavioral specificity $\rho = 0.45$, scope boundaries 0.43, dependency context 0.40, outcome clarity 0.39) and three weakly (error states 0.22, data contracts 0.19, acceptance criteria 0.17). The LLM reliably detects whether specification content is *present* but cannot judge whether it is *correct* for the domain. Since even this completeness-oriented measure shows no defect benefit, the null is conservative.

25,209 PRs (28.6% of total) received quality scores across all 119 repositories. Scoring targeted spec'd PRs whose combined issue and description text exceeded 50 characters; 24,194 of the 24,298 spec'd PRs met that threshold, and a further 1,015 PRs scored under an earlier revision of the classifier were later reclassified as unspecified but retain their scores (excluding them leaves H3 and H4 unchanged, at $p = 0.168$ and $p = 0.829$). For PRs linked to GitHub issues, we fetched the issue body and scored the combined content; 17,094 scored PRs include linked issue content, and the remainder were scored on PR description alone. This subsample is non-random – it is restricted to PRs carrying substantial specification text.

A further limitation: we cannot distinguish specifications written before implementation from descriptions written after. Linked GitHub issues are the strongest available proxy for pre-implementation intent; we test this subset separately in Section 5.6.12. Quality scores for issue-linked PRs (median 50, $N = 13,081$) differ only trivially from description-only PRs (median 47, $N = 12,128$; $p < 0.001$ but a 3-point difference). The null result holds for both subsets.

4.2.3 Defect Introduction (szz_buggy)

We applied the basic SZZ algorithm [22] to 103 of 119 repositories, tracing 64,805 blame links from fix commits to bug-introducing commits. The remaining 16 repositories had unreachable merge SHAs (typically from squash-merge workflows). da Costa et al. [7] found that basic

SZZ misattributes 46–71% of bug-introducing changes. If this noise is non-differential with respect to specification presence, it attenuates associations toward the null rather than creating spurious reversals; Section 5.6.1 bounds both that case and the differential one. A PR is marked `szz_buggy = True` if any of its commits were identified as bug-introducing. 9,754 PRs (12.5% of SZZ-covered repos) are marked as bug-introducing.

4.2.4 Rework (`reworked`)

A pull request is classified as reworked if a subsequent PR within 30 days by any author modifies overlapping files and has a title or description indicating correction (matching patterns such as “fix,” “revert,” “bugfix,” “hotfix,” “regression,” “broke,” or “broken”). File overlap is computed by comparing the set of files changed in each PR pair; high-frequency files (touched by >30% of PRs in the repository) are excluded to avoid spurious attribution. 11,820 PRs (13.4%) are classified as reworked. The 30-day window was chosen based on sensitivity analysis: rework detection increases from 14 to 30 days but shows diminishing returns beyond 30 days (the spec-rework association is directionally consistent across 14-, 30-, 60-, and 90-day windows).

4.2.5 AI-Tagged (`ai_tagged`)

A pull request is classified as AI-assisted by two detection layers. A regex matches co-author tags (Co-authored-by: Copilot), generation attribution strings (Generated with, Claude Code), agent markers (CURSOR_AGENT, coderabbit.ai, AI-generated), and the robot emoji; separately, we parse the structured “AI Disclosure” and “AI Usage” sections some repositories include in PR templates. Validated against 537 PRs with clear self-reported disclosures, the combined classifier achieves 98.8% precision (1 false positive) but 24.2% recall, because many developers describe AI use in natural language our patterns do not capture. 5,989 PRs (6.8%) carry AI tags after bot exclusion – a lower bound.

4.2.6 Code Churn (`total_churn`)

Total code churn is defined as additions + deletions. We use $\log(1 + \text{churn})$ in regressions to normalize the heavily right-skewed distribution (median: 55, mean: 698).

4.3 Statistical Approach

For each hypothesis, we report three levels of analysis. Each level controls for more potential confounders than the last, so readers can see how the result changes as we account for more alternative explanations:

Pooled. Raw rates compared directly. This is the analysis that unadjusted claims would cite – it ignores all confounders.

Controlled. Logistic regression adjusting for PR size (log additions, log deletions, log files changed) and *repository fixed effects*, so the model compares spec’d and unspec’d PRs *within the same repo*. In two analyses (H1, H3), including 100+ repository adjustments caused singular matrices, so those estimates use size controls only.

■ **Table 3** Sample sizes by hypothesis.

Hypothesis	PRs (pre-filter)	Within-author N	Ident. authors
H1: specs $\rightarrow$ bugs	77,814 (103 SZZ)	72,046	2,181
H2: specs $\rightarrow$ rework	88,052 (all 119)	81,647	2,524
H3: quality $\rightarrow$ bugs	22,672 (scored, SZZ)	19,688	2,030
H4: quality $\rightarrow$ rework	25,209 (scored subset)	21,881	2,297
H5: spec $\times$ AI $\rightarrow$ churn	88,052 (all 119)	71,932	366

Within-author N is after restricting to authors with ≥ 2 PRs and dropping rows with missing controls. Identifying authors are those with variation in both treatment and control conditions.

Within-author. The strongest test. We subtract each developer’s own average from every variable, then run a linear regression on the residuals. This compares each developer’s spec’d PRs to their own unspec’d PRs, eliminating all stable differences between developers – skill, experience, coding style, and organizational context are held constant because both “treatment” and “control” come from the same person.¹ Standard errors account for the fact that PRs from the same author are correlated (clustered at the author level). Primary models use size controls; Section 5.6 confirms results hold when adding Just-In-Time (JIT) defect prediction features [13] – a set of 14 code-change metrics (lines added, file age, developer experience, etc.) known to predict defect-introducing commits.

The effect is identified only from authors who have *both* spec’d and unspec’d PRs; those who always or never write specs contribute no information. We therefore restrict to authors with ≥ 2 PRs and report the number with treatment variation throughout, along with 95% confidence intervals from clustered standard errors, so readers can judge which effect sizes are compatible with the data.

We do not apply multiple-comparison corrections across the five hypotheses; at $\alpha = 0.05$, approximately 0.25 false positives are expected by chance, and we interpret all results with that in mind.

Sample sizes vary across hypotheses because of SZZ coverage, quality scoring, and within-author filtering:

5 Results

Our outcomes differ in how heavily they depend on the SZZ algorithm, and we weight them accordingly. Rework (H2) is derived from subsequent pull request activity and does not use SZZ at all. Propensity-matched defect rates compare spec’d and unspec’d PRs with equivalent just-in-time risk profiles. This neutralizes a mechanical artifact: larger PRs touching older files collect more `git blame` hits regardless of their true defect rate. These two are our primary evidence. Within-author estimates on unmatched SZZ outcomes are reported alongside them, and are informative about how the association behaves as controls accumulate, but they are secondary: as Section 5.6.1 sets out, SZZ noise on this particular treatment is unlikely to be non-differential.

¹ Technically: a linear probability model (LPM) with full author demeaning, equivalent to author fixed effects via the Frisch–Waugh–Lovell theorem. We use LPM rather than logistic regression because author demeaning is exact for OLS but produces biased estimates in logit due to the incidental parameters problem [17]. Chamberlain’s conditional logit avoids this but requires discarding groups without outcome variation, reducing power. We follow Angrist and Pischke’s recommendation [2].

■ **Table 4** H1: Specifications and defect-introduction rates, ordered from most to least adjusted. The propensity-matched row is the primary estimate; Section 5.6.9 gives matching detail and balance diagnostics.

Method	Coefficient	<i>p</i> -value	Interpretation
Propensity-matched (primary)	−0.006	0.133	No difference
Within-author + JIT controls	+0.006	0.229	Not significant
Within-author, size controls	+0.014 [+0.005, +0.024]	0.003	+1.4pp, no cplx. adj.
Logistic (size only [†])	0.067	0.006	Log-odds, not pp
Pooled (raw comparison)	OR = 1.20 ^a	< 0.001	Higher defect rate

^a Odds ratio: spec'd PRs are 1.20× as likely to introduce a defect as unspec'd PRs. [†] Including a separate adjustment for each of 100+ repositories caused the model to fail; this estimate adjusts for PR size only. Within-author 95% CI in brackets.

5.1 H1: Specifications Reduce Defects

Analysis restricted to 77,814 PRs in the 103 repositories with SZZ coverage. Table 4 reports estimates from most to least adjusted; the propensity-matched estimate is the primary result.

Within-author estimates are identified from 2,181 authors with treatment variation (out of 4,328 authors with ≥ 2 PRs).

H1 is not supported. After matching on just-in-time risk profiles, spec'd and unspec'd PRs are indistinguishable in defect-introduction rate (−0.6pp, $p = 0.133$, 17,375 matched pairs, with all 16 covariates – the 13 usable JIT features plus the three size controls – balanced). No protective effect appears at any level of analysis.

Reading up the table shows why the raw association exists. It is positive and strongly significant unadjusted, and it weakens monotonically as task complexity is controlled: within-author with size controls gives +1.4pp ($p = 0.003$), adding JIT risk features drops the coefficient by 55% and removes significance ($p = 0.229$), and matching eliminates it. This is *confounding by indication* – developers write specs for their hardest work, and harder work produces more defects.

The complexity-unadjusted +1.4pp estimate should not be read as evidence that specifications *cause* defects, for a reason specific to SZZ. Spec'd PRs add 1.8× more lines and touch 2.3× older files, so they present more `git blame` targets and accumulate defect attributions mechanically, independent of their true defect rate. This biases the unmatched estimate upward. It is one reason we treat the matched estimate as primary, and one reason H2 – which does not depend on SZZ at all – carries weight. Section 5.6.1 bounds both failure modes.

5.2 H2: Specifications Reduce Rework

Analysis includes all 88,052 PRs. Rework is our primary outcome: it is measured from subsequent pull request activity and is independent of SZZ, so the misattribution concerns that qualify H1 do not apply.

H2 is not supported. No specification of this model produces less rework for spec'd PRs. The within-author estimate is in the wrong direction – the same author's spec'd PRs have a 1.2 percentage-point higher rework rate than their unspec'd PRs ($p = 0.001$), about one

■ **Table 5** H2: Specifications and rework rates, ordered from most to least adjusted.

Method	Coefficient	<i>p</i> -value	Interpretation
Propensity-matched	+0.005	0.146	No difference
Within-author + JIT controls	+0.009	0.024	Attenuated; more rework
Within-author, size controls	+0.012 [+0.005, +0.019]	0.001	+1.2pp <i>increase</i>
Logistic (size + repo)	0.178	< 0.001	Log-odds; persists
Pooled (raw comparison)	OR = 1.18 ^a	< 0.001	Higher rework rate

^a Odds ratio: spec'd PRs are 1.18× as likely to be reworked as unspec'd PRs. Within-author estimates identified from 2,524 authors with treatment variation (out of 4,892 with ≥ 2 PRs).

extra rework event per 83 PRs – and matching on task complexity removes the difference rather than reversing it (+0.5pp, $p = 0.146$). The claimed benefit is absent whether or not complexity is adjusted for, and this outcome does not rely on SZZ.

5.3 H3: Specification Quality Reduces Defects

H1 and H2 tested specification *presence*. H3 and H4 test specification *quality* – whether, among PRs that already have specifications, better-written ones predict better outcomes. This is the stronger version of the SDD claim: not just “write a spec” but “write a *good* spec.” Quality is the seven-dimension rubric score from Section 4.2. We treat H3 and H4 as *exploratory* rather than confirmatory, because the measure has limited construct validity: human–LLM agreement is only moderate, and the rubric captures *formal* completeness rather than *functional* correctness – a specification can score highly on every dimension and still specify the wrong behavior.

Of the 22,672 quality-scored PRs in SZZ-covered repos, 19,688 remain after restricting to authors with ≥ 2 PRs and dropping rows with missing controls, identified from 2,030 authors with treatment variation.

H3 is not supported. Adjusting for PR size alone, the quality coefficient is -0.002 ($p = 0.080$), which approaches significance but corresponds to a negligible effect: a 10-point quality improvement predicts a 2 percentage-point defect reduction.² The within-author estimate is an order of magnitude smaller and not significant, -0.0003 (95% CI $[-0.0007, +0.0001]$, $p = 0.164$), implying that a 10-point improvement predicts a 0.3pp reduction. The quality signal does not survive stronger controls, and is non-significant across all robustness checks (Section 5.6).

5.4 H4: Specification Quality Reduces Rework

H4 is not supported. Specification quality has no meaningful relationship with rework. The within-author coefficient is effectively zero, $+0.000$ (95% CI $[-0.000, +0.000]$, $p = 0.860$), and the controlled logistic regression adjusting for size and repository is likewise non-significant (-0.000 , $p = 0.920$). Estimates are identified from 2,297 authors with treatment variation, the same scored subsample as H3.

² Including a separate adjustment for each of 100+ repositories caused this model to fail, so the estimate adjusts for PR size only.

5.5 H5: Specifications Constrain AI Scope

Analysis uses log code churn (additions + deletions) as outcome. PRs with zero churn are excluded (these represent missing data rather than true zero-change events). AI detection combines regex pattern matching with structured AI disclosure parsing (Section 4.2), identifying 5,989 AI-tagged PRs – $2.3\times$ more than co-author tags alone.

Within-author, specifications accompany larger PRs in both populations: $+0.120$ for AI-tagged PRs ($p = 0.030$, identified from 224 authors with treatment variation) and $+0.140$ for human-authored PRs ($p < 0.001$). The spec $\times$ AI interaction is $+0.000$ (95% CI $[-0.101, +0.102]$, $p = 0.997$, identified from 366 authors).

H5 is not supported. Rather than constraining scope, specifications accompany *larger* PRs for both AI-tagged ($+12.0\%$) and human-authored ($+14.0\%$) work. The interaction term is effectively zero ($p = 0.997$): specifications have the same relationship with code churn for AI and human PRs alike – the same confounding-by-indication pattern seen in H1 and H2. Harder tasks get specs and produce more code.

5.6 Robustness Checks

Twelve additional analyses test whether the null result is an artifact of our primary measures or insufficient controls.

5.6.1 SZZ Misattribution Sensitivity

da Costa et al. [7] report that basic SZZ misattributes 46–71% of bug-introducing changes. Because SZZ underpins H1, we bound the consequences in both directions.

If misattribution is *non-differential* with respect to specification presence, it attenuates the observed coefficient toward zero. Treating the reported rates as an approximate attenuation factor, the within-author estimate of $+0.014$ [$+0.005, +0.024$] corresponds to a disattenuated interval of roughly [$+0.9, +4.4$]pp at 46% misattribution and [$+1.7, +8.3$]pp at 71%. Both intervals remain entirely positive. Non-differential noise cannot convert a positive association into the protective one SDD vendors claim; it can only make a positive association look smaller than it is. These are order-of-magnitude bounds rather than estimates: da Costa et al. report a misattribution rate, not a decomposition into sensitivity and specificity, so the correction is approximate and we do not carry the disattenuated intervals into any results table.

If misattribution is instead *differential*, the attenuation argument does not apply. We consider differential noise likely here, because spec'd PRs add $1.8\times$ more lines and touch $2.3\times$ older files and therefore present more `git blame` targets; this would bias the unmatched SZZ estimate upward. Propensity score matching addresses precisely this mechanism – it matches on size, file age, change entropy, and developer experience – and returns a null (Section 5.6.9). Rework (H2) provides an independent check that does not use SZZ at all, and shows the same absence of benefit.

The two failure modes thus have a common answer, and neither yields support for a protective effect. Under non-differential noise the association stays positive; under differential noise it disappears once complexity is matched.

5.6.2 Alternative Outcome Measures

We tested two additional outcomes beyond SZZ-traced defects and rework: *escaped defects* (PRs whose CI passed but were subsequently followed by a fix or revert) and *strict escaped* (escaped PRs where the follow-up explicitly indicates a fix or regression). Both are directionally positive (specs accompany more escapes, not fewer) but not significant (escaped: $p = 0.333$; strict escaped: $p = 0.108$; within-author LPM, $N = 81,647$). No outcome measure shows specifications reducing defects at $p < 0.05$.

5.6.3 Incremental Validity Beyond JIT Features

We tested whether specification information adds predictive power beyond the 14 JIT defect prediction features [13] on 61,192 PRs with complete JIT features and SZZ outcomes. Spec presence is statistically significant when added to the JIT model ($p = 0.003$) but adds negligible predictive power ($\Delta R^2 = 0.0002$). Spec quality adds nothing ($p = 0.142$). JIT features account for nearly all the predictive power; knowing whether a PR has a specification tells you almost nothing beyond what the code-change metrics already reveal.

5.6.4 Quality Dimensions: Individual and Validated Subsets

We tested each of the seven quality dimensions independently against SZZ bugs and rework (within-author LPM, 19,688 PRs). No individual dimension is significant for defects at $p < 0.05$; every coefficient rounds to zero. Scope boundaries ($p = 0.053$) and behavioral specificity ($p = 0.083$) are closest, but neither survives Bonferroni correction for seven simultaneous tests ($p < 0.007$ required). No dimension shows any relationship with rework.

To test whether measurement noise in weakly validated dimensions attenuates the quality signal, we compared composites restricted to the four dimensions with significant human–LLM agreement ($\rho = 0.39$ – 0.45 : behavioral specificity, scope boundaries, dependency context, outcome clarity) against composites using the three unvalidated dimensions ($\rho = 0.17$ – 0.22 : acceptance criteria, error states, data contracts).

All three composites are non-significant for defects: all seven dimensions together give -0.0003 ($p = 0.164$), the four validated dimensions alone -0.0002 ($p = 0.184$), and the three unvalidated dimensions alone -0.0003 ($p = 0.158$). The validated and unvalidated composites therefore perform almost identically, which is what one would expect if the quality score were capturing noise rather than a real defect-predictive signal. For rework the three composites give $+0.0000$ ($p = 0.860$), $+0.0000$ ($p = 0.913$), and $+0.0001$ ($p = 0.663$) respectively – all indistinguishable from zero.

5.6.5 Repo-Level Analysis

Aggregating to the repository level ($N = 98$ repos with ≥ 50 PRs), specification rate has no relationship with either defect rate (Spearman $\rho = -0.11$, $p = 0.27$) or rework rate ($\rho = -0.05$, $p = 0.64$). Repos that write more specs do not have fewer bugs or less rework.

5.6.6 Subgroup Analysis

SDD tools are marketed primarily for AI-assisted workflows. We test whether the null result holds across human-only PRs, AI-tagged PRs, and repos stratified by AI adoption rate. As in all analyses, bot PRs are excluded.

■ **Table 6** Subgroup analysis: specification effects across populations.

Subgroup	→ bugs coef	<i>p</i>	→ rework coef	<i>p</i>
All non-bot PRs (88,052)	+0.014	0.003	+0.012	0.001
Human-only, no AI (82,063)	+0.015	0.003	+0.011	0.004
AI-tagged, non-bot (5,989)	+0.011	0.424	+0.010	0.399
Zero AI-tagged repos (3,562)	−0.016	0.377	−0.033	0.014 [†]
Low-AI repos (51,877)	+0.019	0.001	+0.009	0.061
High-AI repos (32,613)	+0.010	0.185	+0.018	0.003

[†] Nominal *p*-value. This is one of six subgroup tests; under Bonferroni correction the threshold is $\alpha = 0.0083$, which this result does not meet. Row counts apply to the rework columns; defect columns use the smaller SZZ-covered subset (77,814; 72,490; 5,324; 1,596; 45,007; 31,211). The zero-AI row shrinks most, so its defect estimate is the least precise here – but the result discussed below is its *rework* estimate, on the full 3,562.

One label needs defining precisely. “Zero AI-tagged repositories” means repositories in which no pull request carries a marker our classifier detects. Since that classifier has 98.8% precision but only 24.2% recall (Section 4.2), the label means *no detectable AI use*, not *no AI use*. Undetected AI-assisted work is expected in these repositories, and we have no way to bound how much. Any claim we make about this subgroup is a claim about repositories where AI use is invisible to our instrument, not about AI-free development.

One subgroup runs the other way. In repositories with no detected AI use, specifications are associated with *less* rework (−3.3pp, nominal $p = 0.014$). It is the only estimate in the dataset pointing in the direction the SDD rework claim predicts, and we are deliberately cautious about it. It rests on a small subsample (3,562 PRs); it is one of six subgroup tests, and does not meet the Bonferroni-corrected threshold of $\alpha = 0.0083$; and, as Section 4.1.1 notes, our sample size was not fixed in advance, which inflates false positives rather than false negatives and so bears on this result more than on any other in the paper. It is consistent with weak protection, but it is not significant under correction, and we do not build on it.

Two further observations qualify it. The subgroup is defined by absence of *detected* AI use, so it is not a clean human-only stratum. And if the effect were real, its location would still be awkward for the vendor claim: a protective association appearing where AI is invisible, and not where AI is tagged, is the opposite of the pattern one would expect if specifications were *most* valuable for AI-assisted work.

For all other subgroups, the pattern holds. For human-authored PRs, specs are associated with *more* defects and *more* rework – the confounding-by-indication pattern. For AI-tagged PRs, the coefficients are positive but not significant ($p = 0.42$ for defects, $p = 0.40$ for rework). The null holds whether a repo has low or high AI adoption.

5.6.7 High-Quality Specs Only

A likely reviewer objection is that our *specd* measure is too loose – that linked Jira tickets are not “real” specifications in the SDD sense, and only high-quality structured specs should show the benefit. We test this directly by restricting to top-quartile (quality score ≥ 58) and top-decile (≥ 66) specs. Quality thresholds are computed from the scored subset’s distribution (median = 48, $p75 = 58$, $p90 = 66$).

No quality threshold produces a significant defect reduction. The top-quartile result (−0.7pp, $p = 0.190$) and top-decile result (−0.5pp, $p = 0.545$) are both non-significant – the opposite of a dose-response gradient. The AI + high-quality spec result ($p = 0.790$), the closest test to the agentic SDD workflow vendors are selling, shows no signal.

■ **Table 7** High-quality specification thresholds vs. defect rates.

Test	→ bugs coef	<i>p</i>	Ident. authors
Top-quartile specs vs. all others	−0.007	0.190	1,370
Top-decile specs vs. all others	−0.005	0.545	751
High-quality vs. no spec (low-quality excluded)	−0.006	0.348	1,117
AI-tagged + high-quality spec	−0.005	0.790	175

We also test a three-tier dose-response, splitting PRs into three mutually exclusive groups – high-quality specs (top-quartile, $N = 5,253$), any other spec’d PR (below top-quartile or unscored, $N = 16,505$), and no specification artifact ($N = 56,056$, the reference group) – within-author, with the bug analysis restricted to the 103 SZZ-covered repos. If quality matters, the tiers should show a monotonic gradient. They do not. High-quality specs show a raw defect rate of 12.6%, indistinguishable from the 12.0% of PRs with no spec, while ordinary specs show the highest rate at 14.5%. Within-author, high-quality specs give a non-significant −0.4pp reduction ($p = 0.566$) and ordinary specs a significant +2.0pp *increase* ($p < 0.001$); for rework both tiers are positive (+0.017, $p = 0.006$; +0.010, $p = 0.012$). The pattern is consistent with confounding, not a quality gradient.

5.6.8 JIT Features as Primary Controls

A reviewer concern is that size controls (log additions, deletions, files) are insufficient – the JIT defect prediction framework [13] captures task complexity dimensions (entropy, file age, developer experience) that may confound the specification–defect association. We re-run H1 and H2 with 13 JIT features as additional controls alongside size controls (subsystem experience excluded for zero variance).

For H1, the spec coefficient falls from +0.014 ($p = 0.003$) to +0.006 ($p = 0.229$) – a 55% reduction that also removes significance. For H2 it falls from +0.012 ($p = 0.001$) to +0.009 ($p = 0.024$), a 23% reduction that leaves the estimate significant. A discrepancy with PSM (below), which finds $p = 0.146$ for rework, likely reflects functional form: the JIT regression assumes linearity while matching does not. Either way, the rework association is directionally *positive* under both methods – the direction never supports the SDD claim.

5.6.9 Propensity Score Matching

We pair each spec’d PR with an unspec’d PR having a similar JIT complexity profile³, testing whether confounding by indication explains the association.

All 16 covariates are well-balanced after matching. Across 17,375 matched pairs, spec’d PRs introduce defects at 14.7% against 15.2% for their unspec’d matches, a difference of −0.6pp ($p = 0.133$): matching eliminates the raw association entirely. Across 19,648 pairs, rework runs at 15.2% against 14.7%, a difference of +0.5pp that is likewise not significant ($p = 0.146$). Both associations are explained by the JIT risk profile of the tasks that receive specifications. These are the estimates reported as primary in Tables 4 and 5.

³ Nearest-neighbor matching on propensity scores (predicted probability of being spec’d based on JIT features), caliper = 0.05 SD.

■ **Table 8** Issue-linked specifications vs. other operationalizations (within-author LPM).

Test	→ bugs coef	<i>p</i>	→ rework coef	<i>p</i>
Issue-linked vs no spec	+0.011	0.037	+0.014	< 0.001
AI + issue vs AI + no spec	+0.014	0.436	+0.017	0.192
Human + issue vs human + no spec	+0.012	0.035	+0.013	0.002
Top-20% quality issue vs no spec	−0.007	0.363	+0.019	0.007
Issue-linked vs ticket-only	+0.007	0.637	+0.026	0.025

5.6.10 Temporal Analysis: The SDD Era

We restrict to the most recent three months (January–March 2026), when specification rates are highest (29.3%) and AI adoption most prevalent (8.8%), to test whether the null result is a historical artifact.

In the period of highest SDD adoption, specifications are associated with more rework (+0.016, $p = 0.001$, against +0.012 in the full dataset) and directionally more defects (+0.004, $p = 0.435$, against +0.014). Among AI-tagged PRs in this window, specifications affect neither defects (+0.003, $p = 0.822$) nor rework (+0.016, $p = 0.212$), and the spec $\times$ AI interaction remains null (+0.011, $p = 0.861$). The null result therefore persists in the data most representative of the SDD era. SZZ-traced defect rates for recent months are likely underestimates due to right-censoring, but this affects absolute rates, not the relative spec'd versus unspec'd comparison.

5.6.11 Complexity Stratification

We stratify by the top 20% of three complexity measures (code churn, change entropy, composite JIT risk) to test whether specifications help most on the hardest tasks.

The result is opposite to the SDD prediction. In the top quintile, specifications are associated with substantially *more* defects and *more* rework: +0.040 and +0.051 for churn (both $p < 0.001$), +0.024 ($p = 0.057$) and +0.039 ($p < 0.001$) for entropy, and +0.028 ($p = 0.028$) and +0.039 ($p < 0.001$) for JIT risk. In the corresponding bottom-80% strata every coefficient collapses toward zero and none approaches significance for defects (churn +0.007, $p = 0.114$; entropy +0.000, $p = 0.986$; JIT risk +0.002, $p = 0.713$). Confounding by indication is thus strongest precisely where specifications should help most, and the association disappears on simpler tasks. Specification quality (H3/H4) is null across all strata.

5.6.12 Issue-Linked Specifications

A further objection is that our `specd` measure admits weak signals such as bare ticket identifiers. We therefore restrict to the 18,600 PRs that link to a GitHub issue – the closest available proxy for a pre-implementation specification, and, per Table 2, roughly three quarters of the treatment rather than a narrow subset of it.

Issue-linked specifications show the same pattern: defect associations are positive or null, never protective. Even top-20% quality GitHub issues show no defect reduction ($p = 0.363$). For AI-tagged PRs with issue-linked specs, both defects ($p = 0.436$) and rework ($p = 0.192$) are non-significant.

5.6.13 Summary

The null holds across four outcome measures, JIT controls, matching, SZZ misattribution bounds, seven quality dimensions, PR- and repo-level analysis, six subgroup cuts, five complexity strata, issue-linked specs, quality thresholds, dose-response, and temporal restriction. The only estimate pointing the other way is rework in repositories with no detected AI use (-3.3pp , nominal $p = 0.014$), which does not survive correction for six tests. No specification measure robustly predicts fewer defects under any operationalization.

6 Discussion

6.1 Confounding by Indication

The two hypotheses about specification *presence* follow an identical pattern: unadjusted analysis suggests specifications accompany worse outcomes; within-author analysis attenuates but does not eliminate the reversed association; propensity score matching on JIT features eliminates it entirely, confirming confounding by indication [20]. Developers rationally invest specification effort in proportion to task complexity. The remaining three fail differently rather than following the same arc: specification *quality* (H3, H4) shows no association with either outcome at any level of adjustment, and specifications accompany larger AI-tagged and human changes alike (H5), with an interaction that is effectively zero.

JIT risk profiles confirm this: spec'd PRs have more lines added ($1.8\times$), touch older files ($2.3\times$), and involve more prior developers ($1.3\times$), while their authors have *less* experience (median 64 vs. 116 prior commits) – consistent with specifications being a support mechanism for harder tasks.

The null admits two readings – specifications genuinely have no effect, or a small protective effect is concealed by residual task selection – and observational data cannot separate them. What we can say is that estimates are *directionally reversed* across most tests and that matching eliminates both associations. Any protective effect is smaller than this study can detect.

6.2 Why Specifications May Not Prevent Defects

For a specification to prevent a defect, every link in a long causal chain must hold. The developer must correctly identify the problem, know in advance what will solve it, and know how to execute that solution. They must then describe the solution in natural language precisely enough that the AI does not misinterpret it, and the AI must implement that description without introducing bugs of its own. Finally, the specification must contain no substantial omissions – which requires the developer to specify what they do not yet know they do not know. Each condition is individually uncertain, and the probability of the full chain holding diminishes rapidly as they compound. Defects arise precisely where human understanding is incomplete, and specifications are written by the same humans whose incomplete understanding produces those defects.

SDD vendors conflate *directing* the AI (telling it what to build) with *ensuring quality* (fewer defects, less rework). Specifications may be effective at direction, but an agent faithfully executing a spec will reproduce every gap, ambiguity, and wrong assumption in that spec. Direction is not quality.

6.3 Vendor Evidence

No SDD tool vendor has published empirical evidence for these quality claims. GitHub describes Spec Kit as “an experiment” [8] with no effectiveness data. Third-party evaluations are negative (Section 2).

The sample contains no SDD tool output to measure, established two ways. A text search across all 100,247 collected PRs – titles, descriptions, and changed-file paths – returns zero matches for `.specify/`, `.speckit/`, `.kiro/`, or `spec-kit`. A file-tree audit of all 119 repositories, checking each default branch for the directories these tools create, likewise finds none; we ran this manually at data collection (April 2026) and programmatically in July 2026, with the same result both times.

The audit does find three repositories that exclude SDD tool output from version control: `apache/airflow` and `apache/spark` ignore `.kiro/` (in Airflow’s case among AI-tool exclusions alongside `.claude/` and Cursor), and `grafana/grafana` ignores `.specs` with the comment “Spec-kit files shouldn’t be committed without a major design process.” Where these tools are used at all here, their artifacts are deliberately kept out of the repository and are therefore invisible to any artifact-based measurement, including ours – the concrete reason we test specification artifacts rather than SDD tool adoption (Section 7). Vendors have the data to conduct controlled evaluations; the absence of published results alongside specific quality claims is a gap experiments could close.

We do not argue that thinking about requirements is wasted effort. We argue that the *products* claiming to reduce defects through specification artifacts have not demonstrated that they do.

7 Threats to Validity

7.1 Construct Validity

Three gaps separate what we measure from what SDD vendors claim, and the largest is the distance between a specification artifact and spec-driven development as a practice. Spec Kit and Kiro generate structured documents through a guided workflow that an agent then implements; a pull request referencing a tracking issue is not that, and Table 2 shows such references constitute 97.5% of our treatment. Our result concerns specification artifacts in ordinary pull request workflows and should not be read as a measurement of SDD tool adoption – a claim not merely untested here but untestable in this sample, since no repository commits SDD tool output and the three mentioning such tools do so to exclude their output from version control (Section 6.3). The nearest approximation available is the top-quartile quality subsample, scored on the dimensions the tools prescribe, and it too shows no robust effect.

A second gap concerns artifacts versus processes: if the benefit comes from the thinking rather than the document, our measure is too coarse to detect it. Two features of the design limit this without eliminating it. The within-author comparison holds constant any consistent habit of thought a developer brings to their work, and H3 and H4 test quality *within* spec’d PRs – if deeper thinking yielded better artifacts, those artifacts ought to predict better outcomes. They do not.

Third, we cannot observe agentic workflows directly. Even where an AI-tagged PR carries a specification, we do not know whether it was supplied to the agent as input, written after the fact, or never read. The 175-author subsample combining AI tags with high-quality specifications is the closest proxy we have, and a loose one.

7.2 Internal Validity

Defect detection is noisy, and the noise may not be random with respect to our treatment. SZZ’s high misattribution rate compounds with the fact that spec’d PRs present more `git blame` targets (Section 5.6.1). This is why we do not rest the defect result on unmatched SZZ estimates: matching removes the association entirely, and rework provides an outcome that does not use SZZ at all and shows the same absence of benefit (Section 5.6.1 bounds both cases). Even so, confounding by indication and differential SZZ noise remain difficult to disentangle fully.

Confounding by indication is the primary threat to the causal reading. The within-author design removes stable differences in skill, style, and context, but not task selection *within* a developer – the same person may reserve specifications for their harder work. The JIT-controlled models and matching address this directly, and both return a null for defects. The estimator also assumes an author’s skill and tooling are stable across the window, which over twelve months of rapid change in AI tooling is unlikely to hold exactly.

Three measurement limitations remain. Our quality rubric is LLM-applied with limited human validation and captures *formal* quality – whether error states are described – rather than *functional* quality – whether the *right* ones are; it was also applied only to PRs with substantial description text, producing a non-random subsample. The control group may be contaminated: some PRs classified as unspec’d may have been specified verbally or in an external tool we cannot see, attenuating any true effect toward zero. And AI detection depends on voluntary tagging, with 98.8% precision but 24.2% recall, so roughly three quarters of AI-assisted work is invisible – diluting the H5 comparison and contaminating any subgroup defined by the absence of AI tags.

7.3 External Validity

All 119 repositories are open-source, while SDD tools are marketed principally to commercial teams whose specification practices, review norms, and defect economics differ from those of public projects. Our results speak to open-source pull request workflows and should not be extrapolated to commercial development.

The sample is also purposive rather than random, and Section 4.1.1 sets out how it was built and what follows from that. Repositories were stratified on specification practice and AI adoption and deliberately enriched for strong specification practice, which biases toward finding an effect rather than toward the null we report; but the sample was not drawn from an enumerated population, and results cannot be generalized beyond it.

Finally, most pull requests in the window predate widespread agentic coding tools. Within the 5,989 AI-tagged PRs, specifications affect neither defects ($p = 0.424$) nor rework ($p = 0.399$), but a genuinely agentic workflow – a specification authored as agent input and implemented end to end by the agent – is barely represented here, and we cannot rule out that it would behave differently.

7.4 Conclusion Validity

We test five hypotheses without correcting across them, so roughly 0.25 false positives are expected by chance at $\alpha = 0.05$; within the subgroup analysis we do correct, and the single positive result fails the Bonferroni threshold for six tests (Section 5.6.6). Power does not explain the AI interaction result: identified from 366 authors, the estimate is not merely non-significant but effectively zero ($p = 0.997$) with a tight interval around it.

All analyses are post-hoc, and sample size was not fixed in advance (Section 4.1.1). The twelve robustness checks show the null to be stable across outcome definitions, control sets, subgroups, and estimators, but stability across post-hoc specifications is weaker than a pre-registered confirmatory test – the obvious next step for anyone wishing to settle the question.

8 Conclusion

We tested five SDD vendor claims against 88,052 pull requests across 119 repositories, scoring specification artifacts on the same quality dimensions SDD tools prescribe and comparing each developer’s spec’d work to their own unspec’d work. None of the five hypotheses are supported. Once task complexity is matched, spec’d and unspec’d PRs are indistinguishable in defect introduction, and rework shows no protective effect under any specification of the model. Specification quality predicts neither defects nor rework, and specifications do not constrain the scope of AI-tagged changes. Across twelve robustness checks, the null holds. Specification presence proxies for task complexity, not quality improvement.

A single estimate points the other way: in repositories with no detected AI use, specifications are associated with less rework (−3.3pp, nominal $p = 0.014$). It does not survive correction for six subgroup tests, and we report it as suggestive rather than as a finding.

Three caveats bound these results (Section 7): we test specification artifacts – in practice, references to tracking issues and tickets – not SDD tool output, which no repository in the sample commits; SZZ carries 46–71% misattribution noise, which is why matched estimates and the SZZ-independent rework measure carry the argument; and the sample is purposive and open-source. Whether purpose-built SDD tooling would produce different results is a question vendors can answer with the data they already have.

Specifications may still have value – as auditable records of intent, as task-batching mechanisms for AI agents, and as accountability artifacts that invite review scrutiny. These are real benefits. They are not “fewer defects,” and the evidence does not support the claim that specifications are most valuable when AI implements the code.

References

- 1 GitHub Copilot Academy. Spec-Driven Development, 2025. URL: https://copilot-academy.github.io/workshops/immersive-experience/spec_driven_development.
- 2 Joshua D. Angrist and Jörn-Steffen Pischke. *Mostly Harmless Econometrics: An Empiricist’s Companion*. Princeton University Press, 2009.
- 3 Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development, 2001. URL: <https://agilemanifesto.org>.
- 4 Birgitta Böckeler. Understanding Spec-Driven-Development: Kiro, spec-kit, and Tessler, 2025. URL: <https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html>.
- 5 Barry W. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.
- 6 Marc Brooker. Kiro and the Future of AI Spec-Driven Software Development, 2025. URL: <https://kiro.dev/blog/kiro-and-the-future-of-software-development/>.
- 7 Daniel Alencar da Costa, Shane McIntosh, Weiyi Shang, Uirá Kulesza, Roberta Coelho, and Ahmed E. Hassan. A Framework for Evaluating the Results of the SZZ Approach for Identifying Bug-Introducing Changes. *IEEE Transactions on Software Engineering*, 43(7):641–657, 2017. doi:10.1109/TSE.2016.2616306.

- 8 Den Delimarsky. Spec-Driven Development with AI: Get Started with a New Open Source Toolkit, 2025. URL: <https://github.blog/ai-and-ml/generative-ai/spec-driven-development-with-ai-get-started-with-a-new-open-source-toolkit/>.
- 9 Colin Eberhardt. Putting Spec Kit Through Its Paces: Radical Idea or Reinvented Waterfall?, 2025. URL: <https://blog.scottlogic.com/2025/11/26/putting-spec-kit-through-its-paces-radical-idea-or-reinvented-waterfall.html>.
- 10 Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. Rapid Quality Assurance with Requirements Smells. *Journal of Systems and Software*, 123:190–213, 2017. doi:10.1016/J.JSS.2016.02.047.
- 11 Leigh Griffin and Ray Carroll. Spec Driven Development: When Architecture Becomes Executable, 2026. URL: <https://www.infoq.com/articles/spec-driven-development/>.
- 12 Hua Guo, Yunhong Ji, and Xuan Zhou. Sedeve-Kit, a Specification-Driven Development Framework for Building Distributed Systems. *arXiv preprint*, 2025. doi:10.48550/arXiv.2509.11566.
- 13 Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. A Large-Scale Empirical Study of Just-in-Time Quality Assurance. *IEEE Transactions on Software Engineering*, 39(6):757–773, 2013. doi:10.1109/TSE.2012.70.
- 14 Srinivas Rao Marri. Constitutional Spec-Driven Development: Enforcing Security by Construction in AI-Assisted Code Generation. *arXiv preprint*, 2026. doi:10.48550/arXiv.2602.02584.
- 15 Jakob Mund, Daniel Méndez Fernández, Henning Femmer, and Jonas Eckhardt. Does Quality of Requirements Specifications Matter? Combined Results of Two Empirical Studies. In *Proceedings of the 9th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10, 2015.
- 16 Rich Nasczyniec. How Spec-Driven Development Improves AI Coding Quality, 2025. URL: <https://developers.redhat.com/articles/2025/10/22/how-spec-driven-development-improves-ai-coding-quality>.
- 17 Jerzy Neyman and Elizabeth L. Scott. Consistent Estimates Based on Partially Consistent Observations. *Econometrica*, 16(1):1–32, 1948.
- 18 Deepak Babu Piskala. Spec-Driven Development: From Code to Contract in the Age of AI Coding Assistants. *arXiv preprint*, 2026. doi:10.48550/arXiv.2602.00180.
- 19 Yannick Roy. The Kitchen Loop: User-Spec-Driven Development for a Self-Evolving Codebase. *arXiv preprint*, 2026. doi:10.48550/arXiv.2603.25697.
- 20 Maribel Salas, Albert Hofman, and Bruno H. Ch. Stricker. Confounding by Indication: An Example of Variation in the Use of Epidemiologic Terminology. *American Journal of Epidemiology*, 149(11):981–983, 1999.
- 21 Forrest Shull, Victor Basili, Barry Boehm, A. Winsor Brown, Patricia Costa, Mikael Lindvall, Dan Port, Ioana Rus, Roseanne Tesoriero, and Marvin Zelkowitz. What We Have Learned About Fighting Defects. In *Proceedings of the 8th International Symposium on Software Metrics*, pages 249–258, 2002.
- 22 Jacek Śliwinski, Thomas Zimmermann, and Andreas Zeller. When Do Changes Induce Fixes? In *Proceedings of the 2nd International Workshop on Mining Software Repositories (MSR)*, pages 1–5, 2005.
- 23 Ryan Yanchuleff. From Chat to Specs: A Deep Dive into AI-Assisted Development with Kiro, 2025. URL: <https://kiro.dev/blog/from-chat-to-specs-deep-dive/>.
- 24 François Zaninotto. Spec-Driven Development: The Waterfall Strikes Back, 2025. URL: <https://marmelab.com/blog/2025/11/12/spec-driven-development-waterfall-strikes-back.html>.
- 25 Christo Zietsman. The Specification as Quality Gate: Three Hypotheses on AI-Assisted Code Review. *arXiv preprint*, 2026. doi:10.48550/arXiv.2603.25773.