Venture Capital, Engineering Quality, and Technical Debt *

João Dimas† Lampros Pnevmatikos ‡

May 2, 2026

Abstract

We investigate a novel channel of venture capital value creation: the practices through which startups build and maintain their technology. Drawing on more than 80,000 codebase snapshots from over 500 technology firms, we analyze code-level engineering quality metrics around first VC entry. We also introduce a dollar-denominated measure of technical debt—the future cost of engineering shortcuts—that is, to our knowledge, the first in the finance literature. We find that treated firms reduce their rate of critical deviations from best practices by 24% relative to the mean and lower technical debt by $0.97 per line of code. The number of code defects remains stable despite significant codebase growth, indicating that new code is written to a higher standard. A placebo test around later financing rounds shows no comparable improvements, suggesting that the first institutional investor—rather than VC capital per se—brings technical discipline. The results are robust to matching on pre-treatment growth.

JEL Classification: G24, G34, D82, L86, L26
Keywords: Venture capital, technical debt, software engineering quality, code quality, difference-in-differences

*We are grateful to Margarita Tsoutsoura and Todd Gormley for their invaluable comments and guidance throughout this project. We also thank Julie Zhiyu Fu for their insightful feedback and suggestions. We acknowledge support from the Google Research Program, and Pnevmatikos thanks the Onassis Foundation for support. The authors have no other relationships or financial interests to disclose.

†Olin Business School at Washington University in St Louis.

‡Olin Business School at Washington University in St Louis. Send correspondence to Lampros Pnevmatikos; e-mail: l.pnevmatikos@wustl.edu.

1 Introduction

An important question in the corporate finance literature concerns how venture capitalists contribute to the success of the startups they finance (Gompers et al., 2020; Amornsiripanitch, Gompers, and Xuan, 2019; Hsu, 2004). While venture capitalists are known to select high-potential firms, a growing body of evidence indicates that their post-investment involvement also plays an important role in shaping firm performance—for example, Bernstein, Giroud, and Townsend (2016) show that reductions in the cost of monitoring portfolio companies lead to improvements in firm performance, especially in innovation. Prior work documents that VCs reshape governance structures, leadership, and organizational design, but a fundamental operational margin remains unexplored: do venture capitalists also change the *technical practices* through which startups build their products?

We examine whether institutional entry also professionalizes *technical practice*—the engineering processes that govern product quality and scalability—a margin that, to our knowledge, prior work has not studied directly. Existing evidence on VC-driven professionalization focuses on organizational and governance channels: Hellmann and Puri (2002) document that VC-backed firms are more likely to hire professional CEOs, implement stock option plans, and formalize HR policies, and that these changes occur earlier than in comparable non-VC-backed firms. These are important operational changes in leadership and organizational structure, but they do not speak to whether VCs also reshape the day-to-day development practices that govern product quality and scalability.

For technology startups—a central part of the venture capital ecosystem—long-run value depends not only on strategy and governance, but also on the quality of the underlying software. Architectural choices and development practices drive the accumulation of technical debt (TD)—the future cost of remediating code issues—and can shape a startup's ability to scale reliably, iterate quickly, and manage operational risk as the organization grows (Sculley et al., 2015, 2014).

Despite its centrality in software engineering, technical debt has not yet been studied directly in the finance literature; to our knowledge, this paper is the first to do so. TD is not recorded in financial statements, yet it behaves like an economically meaningful liability: accumulated codebase compromises increase the future resources required to maintain and modify a product, and, if left unaddressed, can compound into higher defect risk and slower feature development (Banker, Liang, and Ramasubbu, 2021; Krasner, 2022; McKinsey and Co, 2020, 2023). In that sense, TD is closer to an off-balance-sheet operating liability than to a transitory engineering inconvenience. For firms whose core assets are software, the magnitude and dynamics of these liabilities can materially affect scalability and operational risk in the post-investment period. At the same time, TD is notoriously hard to measure and compare across firms (Banker, Liang, and Ramasubbu, 2021; Krasner, 2022). Our setting allows us to bring an industry-standard TD concept into an empirical finance design by constructing a consistent, dollar-denominated measure from codebase-level static analysis.

In this paper, we examine whether venture capital investment is associated with changes in

software engineering practices, thereby shedding light on an operational input central to product quality and scalability. We assemble longitudinal codebase data for a large set of open-core technology startups and track engineering quality over time around VC entry, defined as first-time VC financing.

Standard theories of VC contracting and monitoring provide a rationale for why venture capitalists monitor after investment and how such monitoring can change internal operating practices. When entrepreneurial actions are difficult to contract on, VC financing combines monitoring with staged capital provision to mitigate agency problems and preserve the option to terminate funding when milestones are not met (Gompers, 1995). Consistent with this logic, venture contracts allocate control and board rights that facilitate intervention when performance is weak (Kaplan and Strömberg, 2003). Beyond governance, models with double-sided moral hazard imply that investor support is itself value-relevant, motivating a substantive post-investment role for VCs rather than passive capital provision (Casamatta, 2003). This logic naturally extends to software-dependent startups, where VC execution experience can translate into more disciplined engineering processes and higher-quality technical choices.

To estimate the relationship between VC entry and engineering quality, we exploit within-firm changes around this event using a stacked difference-in-differences design, comparing newly VC-funded firms to not-yet-funded startups over the same relative-time horizon. Treated and control firms are statistically indistinguishable on pre-treatment engineering quality across all of our metrics, supporting the comparability of the two groups at baseline. This setting allows us to observe granular engineering outcomes across three core dimensions: maintainability issues, security vulnerabilities, and reliability issues (bugs). The resulting panel offers a frequency and level of detail that are typically unavailable in corporate finance data.

We find that VC entry is associated with economically and statistically significant improvements in defect density across multiple dimensions. In our baseline stacked difference-in-differences specifications, treated firms exhibit nearly 3 fewer critical violations per 1,000 lines of code, statistically significant at the 1% level. Critical violations are the highest-severity defects identified by the automated code analysis (e.g. security vulnerabilities and null-pointer dereferences), and the estimated decline represents a 24% reduction relative to the treated-group mean. Total rule violations decline by 6.9 per 1,000 lines of code, and maintainability issues fall by 6.5 per 1,000 lines of code, both statistically significant at the 5% level.

Moreover, the reduction in maintainability issues is concentrated among high-severity defects, those representing structural complexity and architectural problems that can impede scalability, rather than more minor issues (e.g. stylistic/readability).

To translate these findings into more concrete economic terms, we develop a methodology that converts code diagnostics into a dollar-denominated measure of technical debt, calibrated to industry benchmarks on software liabilities. Applying this framework, we estimate that technical debt declines by approximately $0.97 per line of code following VC entry. Furthermore, the technical debt ratio

(software liabilities scaled by software assets) falls by 10 percentage points.

Event-study plots reinforce these findings: the estimated improvements emerge at the quarter of funding and continue to grow in magnitude through the fifth post-treatment quarter, with no detectable pre-trends in any of the engineering quality metrics. This timing pattern is consistent with a discrete shift in development practices following investor entry rather than selection on firms whose code quality was already improving.

Importantly, these improvements manifest in defect *rates* (violations per unit of code) rather than total defect *counts*. Level regressions on log violation counts are statistically insignificant, while codebase size increases by approximately 14.6%. The combination of significant density improvements and stable absolute counts is consistent with new code being written to a higher standard, reflecting improved engineering practices, rather than retroactive remediation of legacy defects.

We subject this interpretation to three additional tests. First, the arithmetic magnitude of the density decline exceeds what codebase expansion alone can explain: a 14.6% increase in code volume can produce at most a 12.7% mechanical reduction in defect density, yet we observe a 24% decline, nearly twice the upper bound. Second, restricting the sample to firms with stable codebases (bottom quartile of post-treatment growth) *amplifies* the result: critical violations decline by 4.5 per 1,000 lines of code, 55% larger than the full-sample estimate. Third, Poisson pseudo-maximum-likelihood models that control explicitly for code volume confirm a 6.0% reduction in critical violations, significant at the 10% level. All three tests converge on the conclusion that the density improvements reflect genuine changes in engineering practice rather than a mechanical denominator effect.

VC funding is inherently non-random: venture capitalists exist precisely to provide selective financing based on their assessment of a startup's potential. Our identification does not require that funding be exogenous in general; it requires that VCs do not select on the specific engineering quality metrics we study (or engineering potential more broadly). We present substantive evidence that this condition holds. At the last pre-treatment period, the standardized difference between treated and control firms in critical violations per 1,000 lines of code is -0.006 (effectively zero), and all other engineering measures fall well below the conventional 0.25 threshold for meaningful imbalance. A joint F-test of all code quality metrics on the treatment indicator fails to reject the null ($F(5, 14{,}756) = 1.07$, $p = 0.38$), with a within-cohort R^2 that rounds to zero: baseline engineering quality has no predictive power for VC funding conditional on timing. This is unsurprising. Static-analysis metrics of the kind captured by SonarQube are not part of standard VC due diligence; investors evaluate market opportunity, product traction, and financial projections, not automated code quality scores. While selection on unobserved firm characteristics remains a possibility, the near-zero balance on our specific outcome variables makes it unlikely that the improvements we document reflect selection on engineering practices.

Given that VCs do not appear to select firms on the basis of pre-existing code quality, the remaining identification concern is that the observed engineering improvements simply reflect the

relaxation of financial constraints: an infusion of capital that allows firms to hire engineers and invest in refactoring. We address this concern through two complementary exercises.

First, we match treated firms to control firms that exhibit similar pre-treatment headcount growth rates, restricting attention to fast-growing startups that followed comparable trajectories but did not receive venture funding. In this more restrictive sample (approximately 46,000 observations versus 325,000 in the baseline), the point estimates for total violations and maintainability issues are larger than the baseline (-9.3 and -8.7 per 1,000 lines of code, both significant at the 5% level), while critical violations and technical debt retain 90% and 128% of their baseline magnitudes, both marginally significant in the smaller sample. A specification-curve analysis over 72 matching configurations confirms that 100% of critical-violation coefficients are negative, as are the vast majority of coefficients for total violations, maintainability issues, and technical debt.

Second, we implement a stage-specificity test re-centered on later financing rounds (Series B and beyond). Around these later rounds, we observe substantial scaling: both total headcount and software engineering headcount increase sharply, mirroring the dynamics documented around first VC entry. Yet we find no corresponding improvement in engineering quality. Coefficients for critical violations, maintainability issues, and technical debt are small in magnitude, statistically insignificant, and often move in the opposite direction of the Seed/Series A results. This asymmetry, comparable resource injections without comparable quality gains, is difficult to reconcile with a pure financial-constraints explanation and instead suggests that the engineering improvements are specific to the initial installation of VC governance infrastructure.

While the robustness exercises confirm the existence and direction of the effect, they do not identify which component of the VC bundle drives quality improvements. To probe the mechanism, we exploit variation in investor type and post-funding hiring composition. We interact the treatment effect with a syndicate indicator (above- vs. below-median number of co-investors at the first VC deal) and find that firms financed by fewer co-investors experience larger quality improvements than those in larger syndicates, consistent with a monitoring rather than capital mechanism (Kang, Li, and Oh, 2022). As corroborating evidence, the small subsample of angel-backed firms (9 of 225 treated firms) exhibits no measurable quality gains despite scaling at comparable rates, and a separate DiD using angel financing as the treatment event produces insignificant coefficients with the opposite sign to the baseline.

An alternative explanation is that VCs improve code quality by facilitating the hiring of senior technical talent. Using LinkedIn employment data, we test whether firms that hire a technical leader (CTO or VP of Engineering) or that have a high ratio of senior engineering hires experience differential quality improvements. The hiring-composition interactions are uniformly insignificant across all code quality outcomes, and the base treatment effect remains significant in every specification. Taken together, these results point toward a governance-and-process channel rather than either a capital-injection or human-capital-injection mechanism.

Our results contribute to several strands of the literature. First, we extend the VC professionalization narrative from organizational and governance domains to technical engineering practice. While prior work has documented that VCs professionalize leadership, human resources, and board structures (Hellmann and Puri, 2002), we provide evidence that this professionalization also reaches the engineering processes through which startups build their core products. The decomposition between angel-backed and institutionally-backed firms identifies monitoring as the operative channel, distinguishing our contribution from a generic capital-injection story.

Second, we complement the "operational engineering" literature in buyout markets (Bernstein and Sheen, 2016) by showing that a similar dynamic operates at an earlier stage of firm development: the first institutional investor brings not only capital and governance, but also a shift toward more disciplined software development. The stage-specificity test, documenting null quality effects around later financing rounds despite comparable scaling dynamics, provides direct evidence that this operational transformation is specific to first VC entry rather than a generic feature of any capital infusion.

Third, we contribute to the growing literature on intangible assets and software-intensive firms (Eisfeldt and Papanikolaou, 2013; Peters and Taylor, 2017). Although technical debt is a well-established concept in computer science and widely discussed among practitioners, this paper is the first in the finance literature to construct a comparable measure across firms. Our dollar-denominated measure provides a tractable framework for quantifying software liabilities analogous to traditional balance-sheet items, and our decomposition of density improvements into new-code quality versus remediation provides a methodological template that may be useful for future research on technology firms, M&A due diligence, and the valuation of software assets.

The remainder of the paper proceeds as follows. Section 2 describes the data sources, sample construction, and our approach to measuring engineering quality and technical debt. Section 3 presents the stacked difference-in-differences design, reports the main empirical results, and includes our decomposition of the quality improvement and mechanism analysis. Section 4 provides robustness analyses, including matching on pre-treatment growth and a stage-specificity test around later financing rounds. Section 5 concludes.

2 Data & Background

We describe the data sources, sample, and variable construction processes used in our analysis. Our dataset is constructed by merging three primary data sources: **(i)** longitudinal codebase snapshots obtained via the GitHub and GitLab APIs; **(ii)** employee-firm matched data from LinkedIn; and **(iii)** funding and deal data from PitchBook.

A primary challenge in the economic analysis of software assets is the proprietary nature of source code, which limits access to internal stakeholders. To overcome this obstacle, we focus our

analysis on firms utilizing an open-core business model. This setting allows us to observe detailed development histories and technical choices for core software assets.

2.1 The Open-Core Business Model

The open-core model combines the collaborative advantages of open-source software with the revenue potential of proprietary licensing. Under this model, a firm releases its core product under an open-source license, making the source code freely accessible to developers and researchers. The firm retains exclusive rights to enterprise-grade features, proprietary modules, or support services, which it monetizes through subscription or licensing agreements.

A prominent example is Posit (formerly RStudio, Inc.), the company behind the RStudio integrated development environment for the R programming language. While the core software is free and open source, the company generates revenue by licensing enhanced versions for commercial applications. This approach is widespread in the technology industry and accelerates market entry by leveraging community-driven innovation (Fershtman and Gandal, 2011; Nagle, 2019). For our purposes, this business model provides a distinctive advantage: it allows direct observation of a firm's development history and technical choices—data typically unavailable to researchers.

The open-core setting also has implications for external validity. Relative to closed-source startups, open-core firms expose substantial portions of their codebase (and often issue trackers) to public scrutiny, which can create baseline quality discipline through reputational concerns, external bug reporting, and community review. If this community discipline substitutes for some of the monitoring and process improvements that VCs bring, the marginal effect of VC entry on engineering quality could be attenuated in our setting and potentially larger in proprietary environments. Conversely, transparency may complement VC involvement by making engineering performance more observable and by facilitating rapid community-driven fixes, so the direction of any external-validity bias is not a priori clear. We therefore interpret our estimates as evidence that VC entry is associated with changes in engineering outcomes in an observable, open-core environment.

2.2 Codebase Data

We define our sample universe using OSSrank, a leading analytics platform for the open-source ecosystem, to identify 649 unique companies that commercialize open-source repositories. These firms primarily operate in the technology sector. We identify 734 distinct software projects associated with these firms.

Using the GitHub and GitLab APIs, we match each project to its repository address—the central storage location for the code underlying a firm's core product. GitHub and GitLab are industry-standard platforms for software development and version control, widely adopted across both startups

and large enterprises. These platforms record every modification to the code as a timestamped "commit" containing rich metadata: the precise nature of the change and the approval history.

We construct a monthly panel of codebases by retrieving the final snapshot of each repository for every month in our sample period (2010–2025). While most repositories are hosted on GitHub, some firms use GitLab. Since both platforms offer nearly identical version control features, we pool these data sources into a combined dataset. We retrieve and analyze complete histories for 639 of the 734 identified projects.[1]

2.3 Funding & Firm Data

To define the timing of VC entry, we use the PitchBook database. For each firm in our sample, we extract the history of investment rounds, specifically observing the deal date, transaction type, investor composition, and funding stage (e.g., Seed, Series A). This allows us to identify the first instance of institutional venture financing. To ensure our sample reflects early-stage venture investment, we exclude firms that enter our data only through "mature" financing events. Specifically, we drop firms whose first observed deal is a public listing, debt issuance, management buyout, acquisition, or late-stage round (Series B or later) with no prior Seed or Series A investment on record. We later reintroduce these excluded firms in our stage-specificity tests.

Since the firms in our sample are predominantly private, early-stage, and venture-backed, standard financial disclosures (e.g., Compustat) are unavailable. To proxy for firm size and growth, we use employee-firm matched data from LinkedIn to construct monthly headcount snapshots.

2.4 Measuring Engineering Quality

We measure engineering quality using SonarQube, a widely used static-analysis tool that scans source code and flags deviations from common coding and security practices. SonarQube is typically run on the current version of a repository during development, whereas our empirical design requires the same measures for past versions. We therefore run SonarQube retrospectively on historical snapshots to construct a firm–month panel of engineering-quality outcomes.

Operationally, for each firm and month in our sample period (2010–2025), we (i) identify the repository state at month-end, (ii) run a SonarQube scan on that codebase, and (iii) store the resulting metrics in a structured panel dataset. Across firms and months, this produces nearly 85,000 distinct codebase snapshots. We store outputs in our own research database because SonarQube Community Edition[2] is designed for monitoring a small number of live projects rather than retaining a large archive of historical scans.

[1] We exclude the 95 remaining projects because their repositories lacked critical files required for consistent SonarQube analysis, or used nonstandard repository structures that prevented successful static analysis in our pipeline.

[2] SonarQube Community Edition is the freely available, open-source distribution and provides the same code analysis functionality as the commercial editions

A major challenge in analyzing diverse open-core repositories is handling multiple programming languages consistently, especially when historical build environments are hard to reproduce. For interpreted languages such as Python, JavaScript, and Ruby, SonarQube analyzes source code directly. For compiled languages such as Java, C, and C++, historical revisions often do not build cleanly, which can break standard analysis workflows. We configure the analysis to operate directly on source files and automatically detect and scan all languages present in each repository.

This source-based approach keeps measurement consistent over time. For example, C/C++ repositories can be scanned without functional build scripts, and Java applications can be scanned from their `.java` files even when compiled `.class` files are missing. We provide detailed specifications for select languages in Appendix B.

This design supports comparability across firms with different technology stacks by applying language-specific rule sets within a uniform measurement framework.

2.4.1 Taxonomy of Engineering Defects

The SonarQube scan checks code against thousands of language-specific rules that reflect widely used software-quality frameworks and industry practice (e.g., ISO/IEC 25010). Each time the code triggers a rule, SonarQube records an "issue"; these issues are the basic unit behind our software quality measures. To make the interpretation concrete, we mention three of the most common Java rule violations highlighted by SonarSource. (i) Commented-out code makes it harder to understand what the software actually does and whether the implementation is complete. (ii) Lingering `TODO` markers are placeholders for unfinished work that often persist and embed a hidden backlog inside the code. (iii) Overly complex functions (high cognitive complexity) are harder to read, test, and modify, raising the cost of future changes and increasing the risk of mistakes.[3]

We group detected issues into three domains. Reliability captures coding errors that can lead to runtime failures or unexpected behavior (e.g., null pointer dereferences, infinite loops). Security identifies vulnerabilities that can compromise the system (e.g., hard-coded credentials, SQL injection risks); we also track *Security Hotspots*, which flag sensitive code segments for manual review. Maintainability captures structural issues that make the code harder to modify or extend (e.g., duplicated code, complex control flow), corresponding to long-run productivity costs and technical-debt accumulation.

Each issue is assigned a severity level based on its potential impact.[4] In the empirical analysis, we focus on *Critical Violations*—the highest-severity issues that pose substantial risk to system stability or security, such as defects with a significant probability of production crashes or major security flaws.

[3] For additional examples of common Java issues highlighted by SonarSource (e.g., duplicated string literals, unused elements, raw types, generic exceptions), see SonarSource, "Top issues in Java projects".

[4] SonarQube's severity taxonomy includes a *Blocker* category above Critical for issues that should be fixed immediately. However, in practice these are mostly false positives—code flagged as catastrophic would rarely pass into production in the first place. We therefore focus on Critical Violations as the effective highest-severity category in our analysis.

We distinguish these from lower-severity violations (*Major, Minor*) to separate non-negotiable defects from incremental quality improvements.

2.5 Technical Debt

One of our primary economic outcome variables is Technical Debt (TD): the stock of maintainability liabilities in a codebase. In the software engineering literature, technical debt captures decisions that speed up short-run development but raise the cost of future changes and maintenance efforts (Sculley et al., 2014). In essence, firms "pay interest" in the form of higher maintenance costs in the future and incur costly "paydowns" once refactoring of the code is unavoidable.

To measure TD, we use SonarQube's SQALE (Software Quality Assessment based on Lifecycle Expectations) methodology (Letouzey, 2012). SonarQube scans each codebase, flags maintainability issues (e.g., code duplication, excessive complexity), and assigns each issue a model-implied remediation time. For example, SonarQube might assign 2 minutes to fix a documentation violation, 20 minutes for a minor maintainability issue, and 5 hours for a more severe structural problem. These estimates come from SonarQube's built-in remediation model. Aggregating these remediation times across all issues in a repository-month yields the SQALE Index (measured in hours)[5].

The raw SQALE Index is a useful within-sample measure, but its absolute level is not economically interpretable: it depends on which maintainability rules are enabled and how the remediation-time model weights different issues[6]. To convert the SQALE Index into dollar-denominated technical debt and ensure comparability across firms, we calibrate it to an external benchmark. We apply a single scaling factor, κ, that maps SonarQube's model-implied remediation times into economically meaningful debt levels.

The calibration proceeds in three steps. First, we construct a proxy for the replacement cost of each firm's software assets. Let ℓ denote the development time per (non-comment) line of code, which we set to $\ell = 0.5$ hours (30 minutes)—SonarQube's default parameter. For a codebase with $\text{LoC}_{i,t}$ lines of code, the replacement cost in developer hours is:

$$\text{Re-development Effort}_{i,t} = \ell \times \text{LoC}_{i,t}. \tag{1}$$

Multiplying by the hourly wage of a software engineer ($w = \$55$, U.S. Bureau of Labor Statistics (2025)) yields Software Assets in dollars:

$$\text{Software Assets}_{i,t} = w \times \text{Re-development Effort}_{i,t}. \tag{2}$$

Second, we choose a scaling factor κ that converts the raw SQALE Index into calibrated

[5] SonarQube stores remediation time in minutes; we convert it to hours.

[6] Different quality profiles can therefore produce different totals because rule sets overlap and differ in coverage. To further ensure comparability, we apply a single, fixed profile—SonarQube's default maintainability rule set—across firms and time

remediation effort (in developer hours). Survey evidence suggests that firms typically assess technical debt at 20–40% of their software assets (McKinsey and Co, 2023). We use the conservative lower end of this range and set κ so that the median firm's technical debt equals 20% of its software assets. Formally, let $\varphi = 0.20$ denote this target ratio. We choose κ such that:

$$\text{Median}\big(w \times \kappa \times \text{SQALE}_{i,t}\big) = \varphi \times \text{Median}\big(w \times \ell \times \text{LoC}_{i,t}\big)\,, \tag{3}$$

where medians are computed across all firm-month observations in the sample. Simplifying, this yields:

$$\kappa = \varphi \times \frac{\text{Median}\big(\ell \times \text{LoC}_{i,t}\big)}{\text{Median}\big(\text{SQALE}_{i,t}\big)}. \tag{4}$$

In our sample, this produces $\kappa = 36.62$, which we hold fixed throughout the analysis.

Third, we apply this calibration to construct Technical Debt for each firm-month:

$$\text{Technical Debt}_{i,t} = w \times \kappa \times \text{SQALE}_{i,t}. \tag{5}$$

Because κ is constant across firms and time, it rescales the SQALE Index proportionally. This leaves the statistical significance of regression coefficients unchanged while making the magnitude economically interpretable.

This approach yields a dollar-denominated measure that is comparable across firms and over time. Because it is computed from static analysis of source code, it can be constructed retrospectively for each monthly snapshot, allowing us to track the evolution of technical debt at high frequency over the 2010–2025 sample period. We construct TD exclusively from maintainability issues, which capture the structural "cleanliness" of the codebase.

2.6 Key Variables

Our empirical analysis uses several categories of variables to measure engineering quality and firm characteristics. The primary SonarQube-derived quality metrics are measured per 1,000 lines of code to enable cross-firm comparisons. *Critical Violations* represent the highest severity defects with significant probability of severe consequences such as production crashes or major security flaws. *Violations (All)* capture total deviations from engineering best practices across all severity levels, spanning reliability, security, and maintainability domains. *Maintainability Issues* measure code patterns that hinder future modification or extension—such as excessive complexity, code duplication, or poor structural organization—and we distinguish between high-severity issues (indicating substantial technical debt accumulation) and low-severity issues (minor readability or structural deficiencies). *Bugs* denote confirmed coding errors that inevitably lead to runtime failures or unexpected behavior.

Reliability Issues capture defects posing risks to system stability, again distinguished by high severity (critical defects affecting core functionality or large user bases) and low severity (minor edge-case flaws). Finally, *Security Hotspots* identify security-sensitive code segments that require manual audit; unlike rule violations (which are confirmed deviations), these represent potential risk areas requiring human review.

In addition to these code-quality metrics, we use the economic measures of Technical Debt and Technical Debt Ratio described in Section 2.5. Technical Debt is normalized by lines of code (TD per line of code), while Technical Debt Ratio scales debt principal by software assets. Our analysis also incorporates firm scale variables: lines of code (LoC) measures codebase size, while total firm headcount and software engineer headcount capture organizational scale. Treatment structure variables include the VC investment indicator ($VC_{i,c,t}$, equal to one if firm i in cohort c has received venture capital financing at or before month t), and factors that allow us to explore heterogeneity in the effects, such as High Violations (codebase critical violation density exceeds the sample median at the time of investment) and Large Round (investment round size exceeds the sample median). See Appendix A, Table 16, for comprehensive definitions of all variables used in our empirical analysis.

2.7 Summary Statistics

Table 1 reports summary statistics for the main firm-month panel used in our baseline analysis.

Table 1: Summary Statistics

		Distribution						
Variable	**N**	**Mean**	**SD**	**Min**	**P25**	**Median**	**P75**	**Max**
Panel A: Raw Count Variables (SonarQube Absolute Values)								
Technical Debt (in $)	47,846	1,765,956.00	7,419,980.00	0.00	18,465.00	207,740.00	1,033,754.00	237,273,196.00
Comment Lines	44,974	22,472.00	63,509.00	0.00	654.00	4,230.00	15,511.00	1,009,374.00
Total Violations	47,846	6,261.00	29,184.00	0.00	81.00	758.00	3,479.00	940,639.00
Critical Violations	47,846	1,756.00	5,154.00	0.00	11.00	183.00	1,180.00	108,124.00
Bugs	47,846	525.00	4,770.00	0.00	1.00	16.00	120.00	164,926.00
Security Hotspots	47,846	160.00	746.00	0.00	2.00	21.00	101.00	52,208.00
Maintainability Issues (Total)	47,846	5,703.00	25,050.00	0.00	69.00	703.00	3,265.00	773,416.00
Maintainability Issues (High)	47,846	1,743.00	5,112.00	0.00	10.00	178.00	1,168.00	105,731.00
Maintainability Issues (Low)	47,846	1,562.00	5,569.00	0.00	11.00	132.00	673.00	71,079.00
Reliability Issues (Total)	47,846	850.00	5,973.00	0.00	2.00	41.00	257.00	182,466.00
Reliability Issues (High)	47,846	13.00	67.00	0.00	0.00	1.00	7.00	2,906.00
Reliability Issues (Low)	47,846	374.00	2,277.00	0.00	0.00	4.00	55.00	35,924.00
Panel B: Per-LoC Variables								
Technical Debt per LoC ($)	44,930	10.98	18.10	0.00	3.39	6.64	13.96	856.84
Violations per 1,000 LoC	44,930	45.53	62.42	0.00	12.18	22.50	59.52	1,111.11
Critical Violations per 1,000 LoC	44,930	13.64	22.44	0.00	1.86	5.86	12.90	333.33
Bugs per 1,000 LoC	44,930	3.22	10.97	0.00	0.07	0.72	2.07	415.49
Security Hotspots per 1,000 LoC	44,930	2.85	16.80	0.00	0.19	0.72	1.66	666.67
Maintainability Issues (All) per 1,000 LoC	44,930	41.86	58.16	0.00	10.92	21.01	54.46	1,111.11
Maintainability Issues (High) per 1,000 LoC	44,930	13.41	22.16	0.00	1.70	5.77	12.68	333.33
Maintainability Issues (Low) per 1,000 LoC	44,930	11.85	25.61	0.00	1.47	4.49	10.23	560.78
Reliability Issues (Total) per 1,000 LoC	44,930	6.44	19.95	0.00	0.35	1.28	4.20	615.94
Reliability Issues (High) per 1,000 LoC	44,930	0.23	2.99	0.00	0.00	0.02	0.11	178.22
Reliability Issues (Low) per 1,000 LoC	44,930	2.98	13.79	0.00	0.00	0.13	0.84	212.31
Panel C: Scale Variables (Log)								
Ln(Lines of Code)	44,974	10.26	2.37	0.00	8.95	10.54	11.96	15.97
Ln(Headcount)	37,056	2.96	1.92	0.69	1.39	2.56	4.04	11.97
Panel D: Treatment Indicators (Post-Funding)								
Post Seed Round (=1)	95,040	0.161	–	0	0	0	0	1
Post Series A (=1)	95,040	0.094	–	0	0	0	0	1
Post Top-Tier VC (=1)	95,040	0.048	–	0	0	0	0	1
Post Mature Round (=1)	95,040	0.105	–	0	0	0	0	1

Notes: Panel A reports SonarQube *raw counts* (absolute values). Panel B reports the corresponding *normalized* measures (units as labeled). Panel C rep

3 Estimation & Empirical Results

To explore the operational consequences of venture capital investment, we estimate the evolution of numerous engineering quality metrics in treated firms relative to controls using a stacked difference-in-differences (D-i-D) framework. We adopt this approach to address the specific challenges posed by the staggered nature of VC financing, where treatment occurs at different times for different firms across the sample. A significant body of econometric research demonstrates that in the presence of dynamic treatment effects, standard two-way fixed effects (TWFE) estimators can be biased. This bias arises because TWFE models implicitly use already-treated units as effective controls for later-treated units, which can yield misleading inferences and, in some cases, produce estimates with the wrong sign (Callaway and Sant'Anna, 2021; Sun and Abraham, 2021). Our stacked design circumvents this

issue by constructing a "clean" counterfactual, ensuring that treated firms are compared strictly to not-yet-treated controls within the same calendar window (Gormley and Matsa, 2011).

We construct our stacked D-i-D event cohorts by identifying the first VC financing event for every treated firm (Seed, or Series A in the absence of Seed). All firms receiving funding within a given calendar month are grouped into the same cohort, while firms that remain untreated by the end of the relevant event window serve as the control group for that cohort. We then stack[7] these cohorts and align them in event time relative to the month of treatment (t=0), observing outcomes over a symmetric 24-month window ($t \in [-12, +12]$). We provide a more detailed derivation of the estimand in Appendix C. This procedure yields 85 unique cohorts, comprising 225 treated firms and 330 controls.

We estimate the baseline relationship using the following specification:

$$M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t} \tag{6}$$

where i, c, and t index the firm, cohort, and time (month). $M_{i,c,t}$ denotes our engineering quality metrics. The key independent variable, $VC_{i,c,t}$, is a treatment indicator equal to one if firm i in cohort c has obtained venture capital financing (first Seed or Series A round) at or before time t, and zero otherwise. Mathematically, this represents the interaction $\text{Treated}_{i,c} \times \text{Post}_{i,t}$, where Treated identifies firms that receive VC funding and Post equals one from the funding month onward ($t \geq 0$) and zero otherwise. The coefficient β estimates the average change in engineering quality for treated firms post-investment relative to controls.

We include Firm × Cohort fixed effects ($\chi_{c,i}$) to absorb time-invariant firm characteristics within a cohort, and Month × Cohort fixed effects ($\delta_{c,t}$) to control for common time-varying shocks affecting the cohort. The vector $X_{i,t}$ represents interactions between the treatment indicator and firm characteristics that allow us to conduct our heterogeneity analysis. Specifically, we include Post × High Violations (where High Violations equals one if a firm's pre-treatment critical violation density exceeds the treated-firm median), Treated × Post × High Violations, Post × Large Round (where Large Round equals one if the VC deal size exceeds the treated-firm median), and Treated × Post × Large Round. The first interaction tests a "room for improvement" hypothesis—whether firms with worse baseline code quality benefit more from VC-induced discipline—while the second tests whether the magnitude of capital investment, rather than monitoring per se, drives the quality improvements. These interaction terms are not absorbed by the fixed effects because they vary within cohort-firm over event time: the Post indicator switches from zero to one after the funding event, generating within-cohort variation that is not absorbed by the Firm × Cohort and Month × Cohort fixed effects. In Appendix F,

[7] As is standard in stacked difference-in-differences designs, the total number of observations exceeds that in the underlying panel because control firm–month observations may appear in multiple cohort-specific sub-experiments when they satisfy the clean-control criterion.

we further explore heterogeneity along founding-team characteristics—whether the founding team includes a technical co-founder (CTO or VP of Engineering) and whether any founder is a serial entrepreneur—using LinkedIn career history data. See Appendix A, Table 16, for comprehensive variable definitions.

Finally, to estimate dynamic treatment effects, we interact the treatment indicator with a set of event-quarter indicators and estimate the evolution of outcomes over the 24-month window. For visual clarity, we aggregate monthly coefficients into quarterly indicators in the event-study figures. The quarter mapping groups three consecutive months into each bin: the omitted reference quarter $q = 0$ contains months $t \in \{-4, -3, -2\}$—all pre-treatment—while the first post-reference quarter $q = +1$ pools months $t \in \{-1, 0, 1\}$. Because $q = +1$ includes one pre-treatment month alongside the first two post-treatment months, any treatment effect that begins precisely at $t = 0$ is attenuated in this bin. The remaining quarters ($q = +2$ through $q = +5$) consist entirely of post-treatment months. This alignment results in event quarters ranging from $q = -3$ to $q = +5$.

3.1 Identification Strategy

We acknowledge that venture capital investment is not randomly assigned; this constitutes a common identification obstacle in the entrepreneurial finance literature. Investors may actively screen for high-potential startups, creating selection bias where treated firms likely differ from untreated firms in unobservable ways (e.g., managerial quality or market potential). A primary concern is that VCs may select firms that are already on an upward trajectory regarding engineering sophistication, or that the investment coincides with other unobserved firm-level shocks. A similar related concern is reverse causality: because VC investments are forward-looking, investors may select firms they anticipate will improve—perhaps because the founders have already committed to a code refactoring initiative or engineering hire that has not yet materialized in observable metrics. Under this interpretation, the post-investment improvements we document would reflect investor foresight rather than investor influence.

However, several features of our data and design attempt to mitigate the concern that our results are driven purely by endogenous selection or omitted variables.

First, our identification relies on the timing of the investment. For unobserved heterogeneity to drive our results, a firm-level shift in engineering quality (such as a decision to refactor the codebase independent of the VC) would need to occur around the funding event and manifest as a discrete change in outcomes at $t = 0$, rather than as a gradual divergence in the pre-treatment quarters.

Second, the "parallel trends" assumption appears to hold in our event studies for the relevant engineering metrics. As shown in the following figures, we observe no significant divergence in engineering measures (normalized by lines of code) between treated and control firms in the quarters

leading up to the investment ($t < 0$). The divergence in engineering quality, specifically the reduction in critical violations, occurs sharply following the deal date ($t > 0$).

Moreover, improvements to code quality are unlikely to be a simple mechanical byproduct of the project's growing scale. If anything, rapid scaling can raise coordination costs and complexity, which would tend to put upward pressure on defect and debt measures even after normalizing by lines of code. Observing declines in these per-LoC outcomes despite concurrent expansion is therefore informative.

We cannot fully rule out reverse causality without a natural experiment. The sharp discontinuity at $t = 0$, combined with the absence of pre-trends, is consistent with outcomes changing at VC entry—whether through active monitoring, resource injection, or professionalization—but it is also consistent with investors successfully forecasting improvements that would have occurred regardless. Our design can rule out selection on *past* engineering quality (since we observe no pre-treatment divergence in code metrics), but it cannot definitively distinguish investor influence from investor foresight regarding *future* quality trajectories.

To address remaining concerns, we implement two additional specifications in our robustness section. First, we estimate a matched Difference-in-Differences model where control firms are paired with treated firms based on pre-treatment headcount growth trajectories. This addresses the concern that our results are driven by comparing VC-backed firms to slower-growing controls, though it does not resolve the forward-looking selection problem. Second, we conduct a stage-specificity test around Series B+ funding rounds. This test is designed to distinguish the effect of venture capital monitoring from the mechanical effect of relaxing financial constraints. If the observed improvements were driven purely by the addition of engineering resources, we would expect similar discontinuities in engineering quality around later-stage rounds. The absence of such effects is suggestive—though not conclusive—that the initial VC financing event is associated with a distinct shift in engineering quality that later capital injections do not replicate.

Baseline balance.

Table 2 reports covariate means for treated and control firms at the last pre-treatment period ($t = -1$). The comparison reveals near-perfect balance on all engineering quality metrics. In the full sample, the standardized difference in critical violations per thousand lines of code is -0.006—effectively zero—indicating that treated and control firms enter the event window with statistically indistinguishable code quality. Similarly, violations per kLoC (-0.128), technical debt per LoC (-0.122), and the SQALE debt ratio (-0.121) all fall well below the conventional 0.25 threshold for meaningful imbalance. Lines of code, our primary scaling denominator, are also closely balanced (std. diff. = -0.030).

The one notable imbalance is in firm scale: treated firms are substantially smaller than controls in total headcount (std. diff. = -0.457) and software engineering headcount (std. diff. = -0.513).

This reflects the composition of our sample, in which treated firms are early-stage startups receiving their first institutional financing while many control firms are more established. Crucially, this level difference is absorbed by the cohort-by-firm fixed effects ($\chi_{c,i}$) in our specification; identification requires only that headcount *trends* evolve in parallel, not that levels be equal. The matched sample, which pairs firms on pre-treatment headcount growth trajectories, narrows some of these differences but the headcount gap persists (std. diff. ≈ -0.5), consistent with the institutional setting in which Seed and Series A recipients are inherently smaller organizations.

The near-zero balance on engineering quality metrics also speaks to a deeper identification concern: selection on code quality. If venture capitalists screened on software engineering practices—selecting firms with either superior or inferior codebases—we would expect systematic baseline differences in these metrics. Instead, critical violations, technical debt, and violation density are virtually identical across groups. This pattern is consistent with the institutional reality that static-analysis metrics of the kind captured by SonarQube are not part of standard VC due diligence. Investors evaluate market opportunity, founding team credentials, product-market traction, and financial projections—not automated code quality scores. The engineering practices captured by our outcome variables are thus unlikely to enter the investor's information set at the time of the funding decision, making selection on these specific outcomes implausible.

We formalize this observation with a linear probability model in which the dependent variable is the treatment indicator and the regressors are all engineering quality metrics measured at baseline ($t = -1$), absorbing cohort fixed effects with standard errors clustered by firm. No individual code quality predictor is statistically significant (all $p > 0.5$), and a joint F-test fails to reject the null that all coefficients are zero ($F(5, 14{,}756) = 1.07$, $p = 0.38$). The within-cohort R^2 rounds to zero, confirming that pre-treatment engineering quality has no predictive power for VC funding conditional on timing. This result reinforces the identifying assumption that our outcome variables are not part of the investor's screening criteria, and that post-investment changes in these metrics can be attributed to the treatment rather than to selection.

Table 2: Baseline Balance: Treated vs. Control Firms at $t = -1$

	Full Sample			Matched Sample		
	Treated	**Control**	**Std. Diff.**	**Treated**	**Control**	**Std. Diff.**
Log(Lines of Code)	10.335	10.398	−0.030	10.449	10.793	−0.162
Log(Headcount)	2.243	3.033	−0.457	2.520	3.413	−0.507
Log(SWE Headcount)	1.459	2.217	−0.513	1.663	2.506	−0.549
Critical Violations / kLoC	12.358	12.481	−0.006	14.224	12.910	0.069
Technical Debt per LoC	7.800	9.006	−0.122	8.833	9.374	−0.050
Violations / kLoC	40.033	47.464	−0.128	44.187	48.032	−0.062
TD Ratio (SQALE)	0.775	0.894	−0.121	0.878	0.930	−0.048
# Obs (firm-cohort-months)	204	13,090		96	1,761	

Note: This table reports covariate means at the last pre-treatment period ($t = -1$) for treated and control firms. The full sample uses all control firms available in the stacked DiD panel. The matched sample retains the nearest-neighbor matched controls. Std. Diff. = (Treated Mean − Control Mean) / pooled standard deviation. Matching is performed on pre-treatment headcount growth rates, as described in Section 4.1. See Appendix A, Table 16, for full variable definitions.

3.2 Impact on Critical Code Defects

We first examine the most severe category of engineering defects: Critical Violations. These are non-negotiable coding errors that pose significant risks to system stability and security. Unlike general technical debt, which can be strategic, critical violations are rarely optimal.

Table 3 shows an economically and statistically significant reduction in these severe defects. Treated firms exhibit 2.92 fewer critical violations per 1,000 lines of code relative to the control group. We observe a similar reduction in total rule violations across all severity levels, which decline by 6.90 per 1,000 lines of code.

Table 3: Difference-in-Differences Estimates for Critical Violations and All Violations

	Critical Violations			**Violations**		
	(1)	(2)	(3)	(4)	(5)	(6)
Treated × Post	-2.923***	-0.963	-4.077**	-6.897**	-6.053	-6.060*
	(1.094)	(0.667)	(1.837)	(2.795)	(3.808)	(3.198)
Post × High Violations (=1)		-1.129**			-2.011	
		(0.521)			(1.717)	
Treated × Post × High Violations (=1)		-4.688*			-2.312	
		(2.373)			(5.366)	
Post × Large Round			1.016			0.636
			(0.920)			(1.581)
Treated × Post × Large Round			1.936			-2.360
			(2.213)			(5.772)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	325,029	321,291	325,029	325,029	321,291	325,029
R^2	0.820	0.823	0.820	0.817	0.816	0.817

This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Treated × Post indicator. Critical Violations are highest-severity SonarQube rule violations per 1,000 lines of code, denoting issues with significant probability of severe consequences such as production crashes or security flaws. Violations (All) denote total deviations from engineering best practices across all severity levels per 1,000 lines of code. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

Figure 1 plots the estimated coefficients of the treatment dummy interacted with quarter-relative-to-treatment indicators. The lack of a pre-trend prior to investment ($t < 0$) provides suggestive evidence against selection bias; VCs do not appear to select firms that were already improving their software quality. The improvement manifests sharply only after the funding date, consistent with a treatment effect.

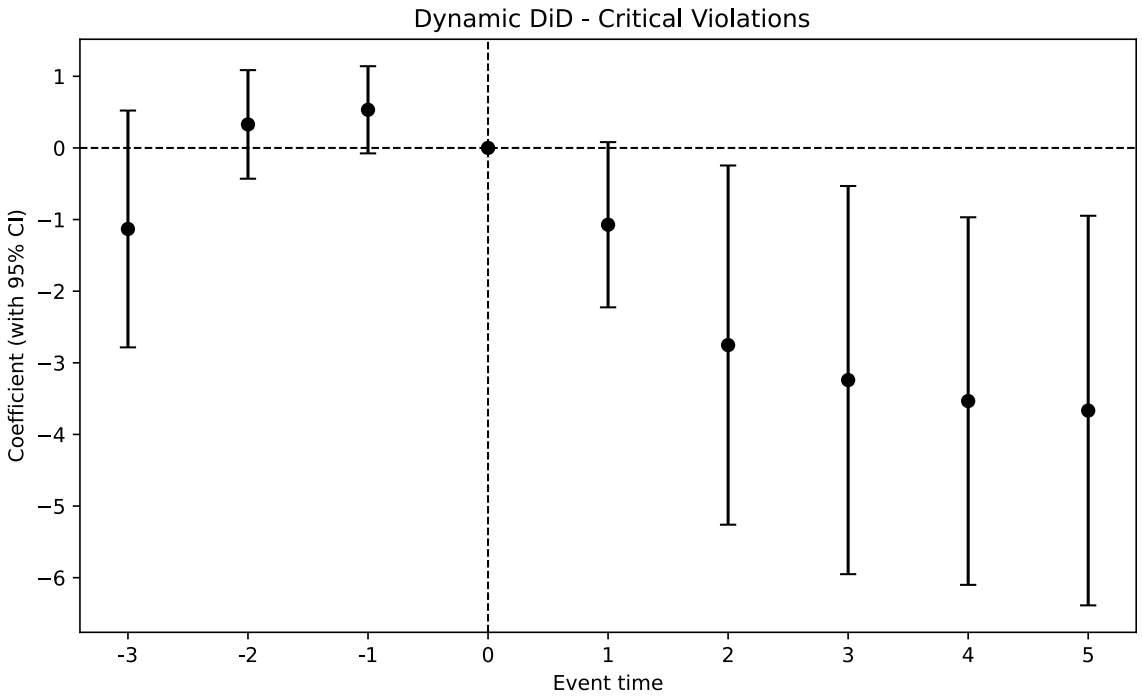


Figure 1: Difference-in-Differences Estimate — # of Critical Rule Violations per 1,000 Lines of Code

We extend this dynamic analysis to the broader measure of all rule violations in Figure 2. While the point estimates for this specification are negative immediately after the deal's quarter, they become statistically significant at conventional levels at $t = 2$ and continue to increase in magnitude through $t = 5$.

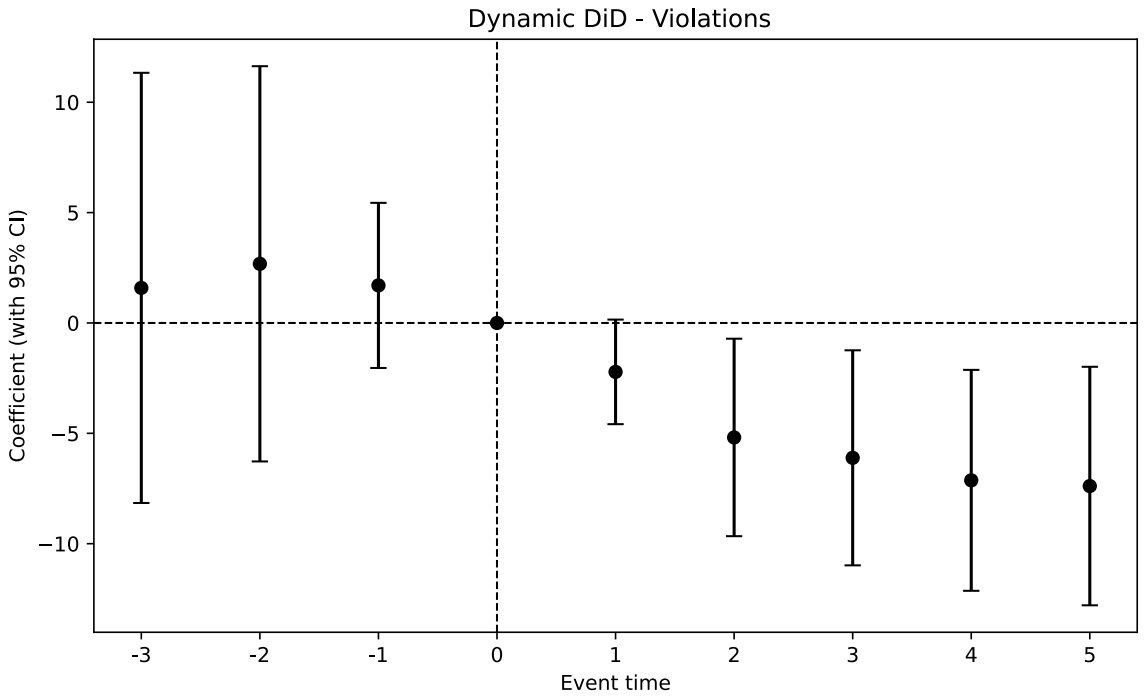


Figure 2: Difference-in-Differences Estimate — # of Rule Violations (All Severities) per 1,000 Lines of Code

3.3 Impact on Maintainability Issues and Software Workflow

We next analyze three maintainability-related engineering quality metrics. Maintainability refers to the ease with which software can be repaired, corrected, or updated. We focus on the change in (1) the total number of maintainability issues across all severities, capturing the aggregate number of code patterns that complicate future modifications or extensions; (2) *High-Severity* Maintainability Issues, which typically capture substantial structural defects such as logical complexity or code duplication that hinder modularity and increase the effort required for updates; and (3) *Low-Severity* Maintainability Issues, denoting readability or other deficiencies that do not impact execution but degrade the code's clarity and organization. These issues are likely to slow down future modification and maintenance.

Table 4 presents the difference-in-differences estimates for Maintainability Issues across varying severity levels. In Column (1), we report an economically and statistically significant reduction in the prevalence of maintainability issues (approximately 6.5 fewer per 1,000 lines of code). This indicates that following VC investment, treated firms exhibit greater adherence to maintainability standards relative to controls.

Decomposing this effect reveals that it is mainly driven by maintainability issues with the largest impact. As shown in Column (4), the reduction is primarily due to *High-Severity* Maintainability Issues—2.7 fewer per 1,000 lines of code—which represent areas of high complexity or architectural issues that inhibit scalability. In contrast, the coefficient for *Low-Severity* Maintainability Issues is

smaller in magnitude and only marginally significant in the baseline specification in Column (7). The disparity in estimates across severity levels suggests that venture capitalists may push the firm to prioritize the resolution of material structural defects. This is consistent with strategically targeting software liabilities that are likely to impede scalability.

Table 4: Difference-in-Differences Estimates for Maintainability Issues

	Maintainability Issues (All)			**Maintainability Issues (High)**			**Maintainability Issues (Low)**		
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Treated × Post	-6.461**	-6.340*	-5.290*	-2.700***	-0.917	-3.784**	-1.059*	-1.161*	-0.691
	(2.621)	(3.785)	(2.715)	(1.019)	(0.664)	(1.688)	(0.609)	(0.697)	(0.580)
Post × High Violations (=1)		-2.124			-1.318***			0.082	
		(1.646)			(0.486)			(0.612)	
Treated × Post × High Violations (=1)		-0.610			-4.304*			0.255	
		(4.950)			(2.205)			(1.127)	
Post × Large Round			0.763			1.055			-0.445
			(1.519)			(0.914)			(0.447)
Treated × Post × Large Round			-3.214			1.749			-0.533
			(5.454)			(2.078)			(0.849)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	325,029	321,291	325,029	325,029	321,291	325,029	325,029	321,291	325,029
R^2	0.790	0.788	0.790	0.822	0.826	0.822	0.813	0.811	0.813

This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Treated × Post indicator. Maintainability Issues (All) capture total maintainability-related rule violations per 1,000 lines of code—code patterns that hinder future modification or extension. High-Severity Maintainability Issues isolate the most severe problems indicating substantial technical debt accumulation. Low-Severity Maintainability Issues capture minor structural or readability deficiencies. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

As in the previous subsection, we plot the temporal evolution of these effects in Figure 3 and Figure 4. Consistent with the assumptions underlying our stacked difference-in-differences design, the estimated coefficients for the pre-treatment period (t<0) are close to zero and statistically insignificant in both specifications. The reduction in maintainability issues manifests following the investment event, corroborating the static estimates presented in Table 4.

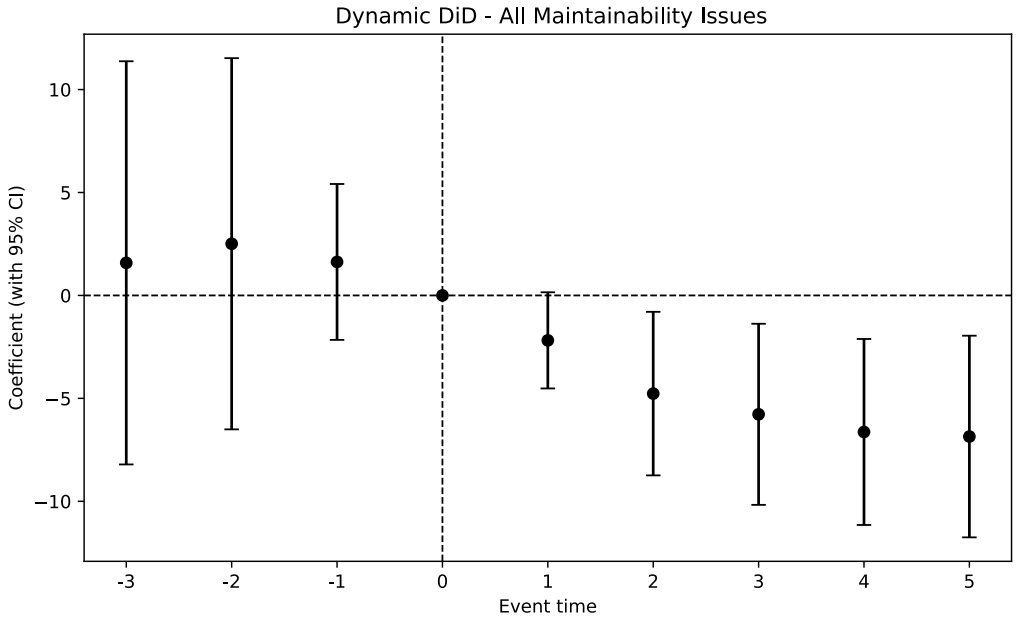


Figure 3: Difference-in-Differences Estimate — # of Maintainability Issues per 1,000 Lines of Code

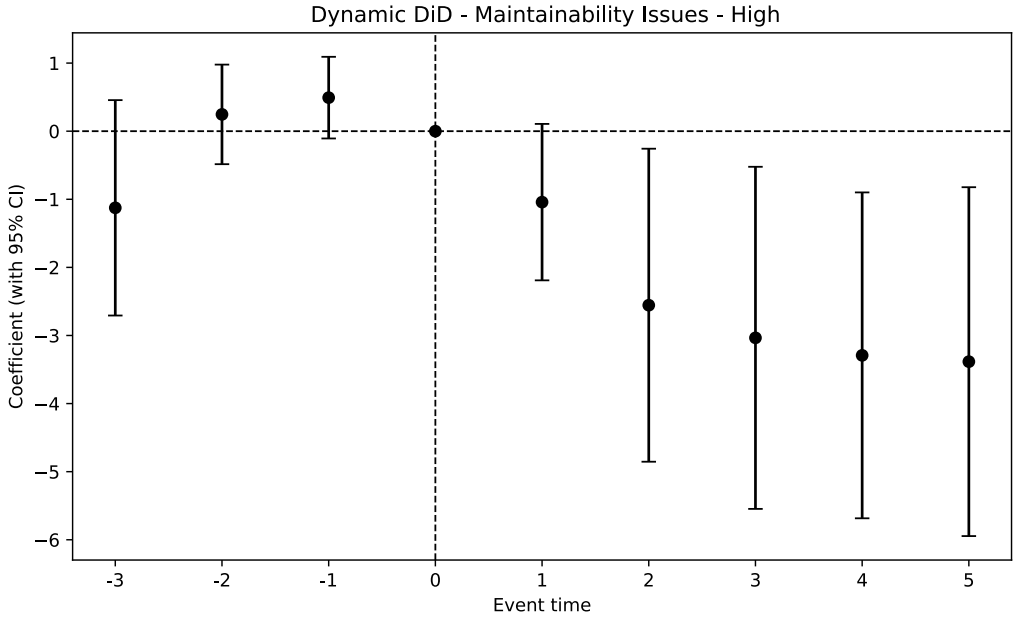


Figure 4: Difference-in-Differences Estimate — # of Maintainability Issues (High-Severity) per 1,000 Lines of Code

To conclude our assessment of software quality, we examine *Security Hotspots*.[8] Security Hotspots identify code segments that are potentially sensitive but require manual audit to confirm

[8]We omit the "Open Issues" metric from the analysis. In retrospective scanning—where the full codebase is analyzed at each snapshot rather than through a live triage workflow—every rule violation is simultaneously an open issue. The two variables are therefore identical in our data, and reporting both would double-count the same information.

vulnerability; unlike rule violations (which are confirmed deviations from best practices), hotspots represent risk areas flagged for human review. Table 5 reports the estimates.

The baseline point estimate for *Security Hotspots* per 1,000 lines of code is negative but not statistically significant (−1.196, Column 1). In the large-round interaction specification (Column 3), the base treatment effect is weakly positive (+0.522, significant at the 10% level), but the triple interaction with large rounds is large and negative, suggesting heterogeneity rather than a uniform increase. Overall, the security hotspot results do not indicate a systematic deterioration in cybersecurity practices post-funding.

Table 5: Difference-in-Differences Estimates for Security Hotspots

	Security Hotspots		
	(1)	(2)	(3)
Treated × Post	-1.196	-2.031	0.522*
	(1.335)	(2.320)	(0.311)
Post × High Violations (=1)		0.496	
		(0.424)	
Treated × Post × High Violations (=1)		2.003	
		(2.307)	
Post × Large Round			0.297
			(0.376)
Treated × Post × Large Round			-4.144
			(2.901)
Cohort × Firm FE	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes
# Obs	325,029	321,291	325,029
R^2	0.532	0.516	0.533

This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Treated × Post indicator. Security Hotspots measure security-sensitive code segments requiring manual audit per 1,000 lines of code; unlike rule violations, hotspots represent potential risk areas requiring human assessment. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

3.4 Impact on Bugs & Reliability

Next, we examine issues that impact the core product's behavior and user experience. Specifically, we analyze the prevalence of software bugs and reliability issues following VC investment.

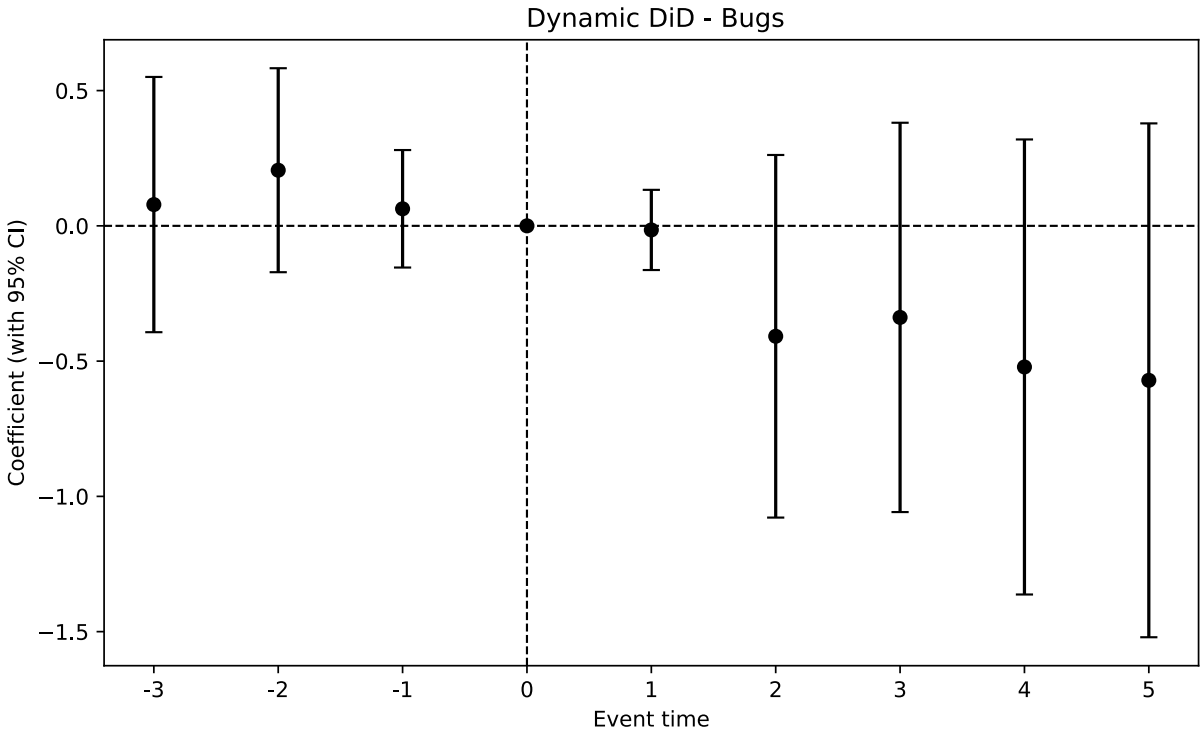
As analyzed in Section 2, Bugs are a specific subset of coding errors that will inevitably lead to runtime failures or unexpected behavior, whereas Reliability Issues are a broader software quality measure that encompasses both these definite bugs and reliability-related risky patterns that might cause errors. When segmented by severity, Reliability Issues (High) capture critical defects that pose a significant risk to system stability and require fast remediation, typically because they are likely to affect a large number of users or core functionality. Reliability Issues (Low) denote minor risks or potential flaws in edge cases that have a negligible immediate impact on the application's runtime performance but may degrade its robustness over time.

Table 6 reports the coefficient estimates. The point estimates for β are negative and economically meaningful (0.45 fewer bugs per 1,000 lines of code), but they are not statistically significant at conventional levels. Nonetheless, these results are directionally consistent with the narrative that firms improve engineering quality following VC funding.

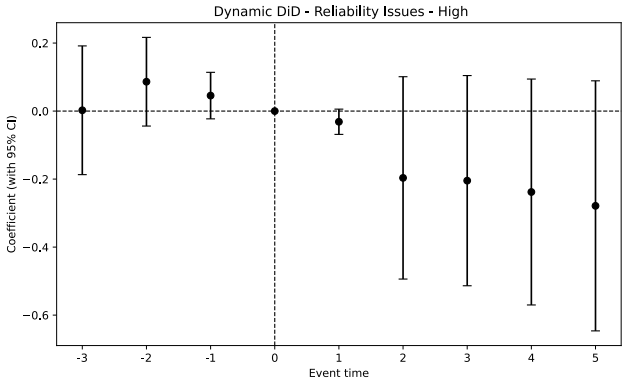
Table 6: Difference-in-Differences Estimates for Bugs and Reliability Issues

	Bugs			**Reliability Issues (High)**			**Reliability Issues (Low)**		
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Treated × Post	-0.459	0.250	-0.785	-0.225	-0.044	-0.291	-0.383	-0.492	-0.431
	(0.389)	(0.269)	(0.593)	(0.153)	(0.037)	(0.232)	(0.399)	(0.698)	(0.451)
Post × High Violations (=1)		0.104			0.191			0.127	
		(0.405)			(0.172)			(0.542)	
Treated × Post × High Violations (=1)		-1.647**			-0.393			0.262	
		(0.751)			(0.298)			(0.718)	
Post × Large Round			-0.157			-0.031			-0.346
			(0.293)			(0.052)			(0.299)
Treated × Post × Large Round			0.856			0.173			0.351
			(0.662)			(0.223)			(0.429)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	325,029	321,291	325,029	325,029	321,291	325,029	325,029	321,291	325,029
R^2	0.833	0.833	0.833	0.740	0.740	0.740	0.797	0.796	0.797

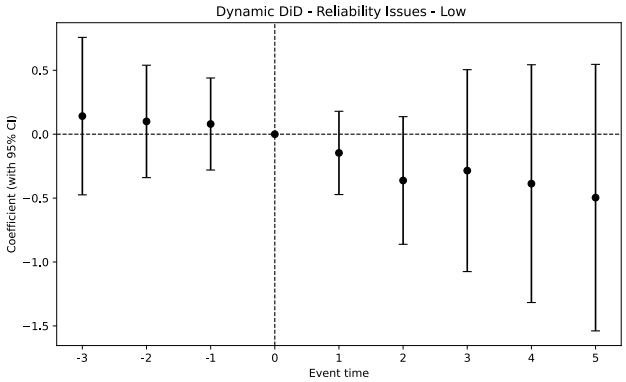
This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Treated × Post indicator. Bugs capture coding errors that inevitably lead to runtime failures or unexpected behavior per 1,000 lines of code. High-Severity Reliability Issues represent critical defects posing significant risk to system stability. Low-Severity Reliability Issues denote minor risks or edge-case flaws with negligible immediate impact. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.



(a) Difference-in-Differences Estimate — Bugs per 1,000 lines of code



(b) Reliability Issues (High Severity) per 1,000 lines of code



(c) Reliability Issues (Low Severity) per 1,000 lines of code

Figure 5: Dynamic Difference-in-Differences Estimates for Bugs and Reliability Issues

3.5 Impact on Technical Debt

Having shown that venture capital investment is associated with a reduction in critical violations and maintainability issues, we now turn to the economically salient measure of Technical Debt. In the computer science literature, Technical Debt (TD) refers to the future cost of reworking code that was originally written quickly to meet short-term goals (Sculley et al., 2014). Companies effectively "pay interest" on technical debt through increased maintenance costs, with eventual costly "paydowns"

required when significant refactoring becomes unavoidable. While technical debt is a widely known concept in engineering, it is rarely discussed in the financial literature.

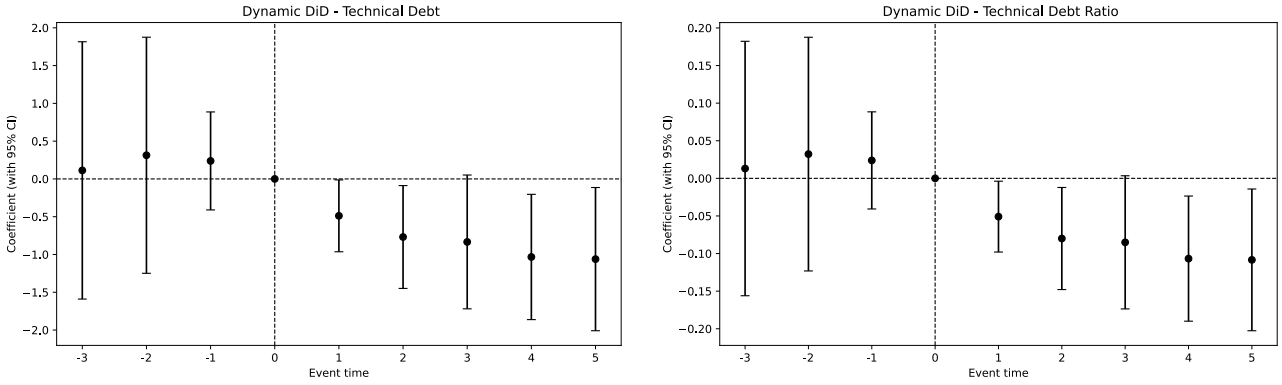
Following our discussion in Section 2, we use the moment-matching calibration to convert SonarQube's remediation time estimates into a dollar-denominated measure of Technical Debt per line of code. We also analyze the *Technical Debt Ratio*, which approximates the firm's software liabilities scaled by the replacement cost of its software assets, as described in Section 2.5.

Table 7 reports the difference-in-differences estimates. Following the first round of VC investment, treated firms exhibit approximately \$0.97 less technical debt per line of code relative to controls. We also find that VC-backed firms reduce their technical debt ratio by approximately 10 percentage points in the post period relative to controls. Relative to the 20% benchmark used to calibrate the technical debt measure in Section 2.5, this change represents a substantial improvement in software capital efficiency.

Table 7: Difference-in-Differences Estimates for Technical Debt

	Technical Debt			**Technical Debt Ratio**		
	(1)	(2)	(3)	(4)	(5)	(6)
Treated × Post	-0.965**	-0.935	-0.854*	-0.099**	-0.094	-0.088*
	(0.460)	(0.664)	(0.467)	(0.046)	(0.066)	(0.047)
Post × High Violations (=1)		-0.380			-0.037	
		(0.356)			(0.035)	
Treated × Post × High Violations (=1)		-0.112			-0.016	
		(0.891)			(0.089)	
Post × Large Round			0.120			0.013
			(0.317)			(0.031)
Treated × Post × Large Round			-0.340			-0.036
			(0.982)			(0.098)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	325,029	321,291	325,029	325,029	321,291	325,029
R^2	0.777	0.776	0.777	0.776	0.776	0.776

This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Treated × Post indicator. Technical Debt per Line of Code and the Technical Debt Ratio are defined in Section 2.5. See Appendix A, Table 16, for variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.



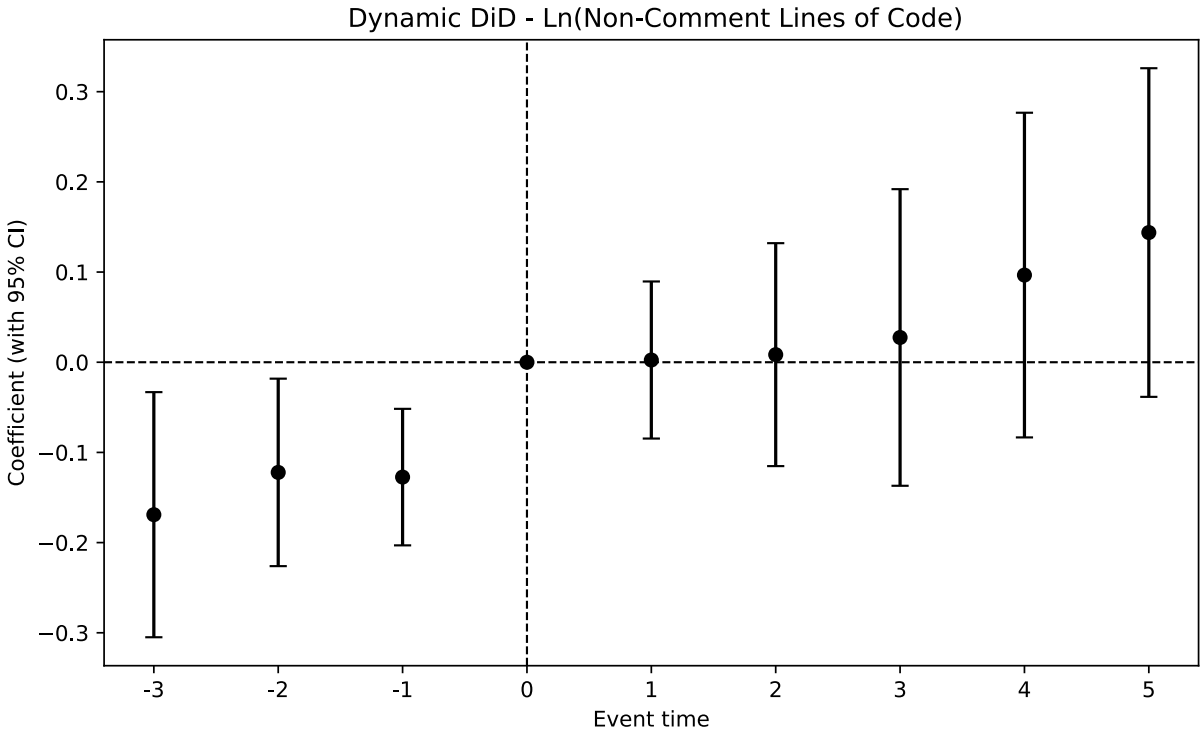
(a) Difference-in-Differences Estimate - $ of Technical Debt per line of code

(b) Difference-in-Differences Estimate - Ratio: Technical Debt over Software Assets

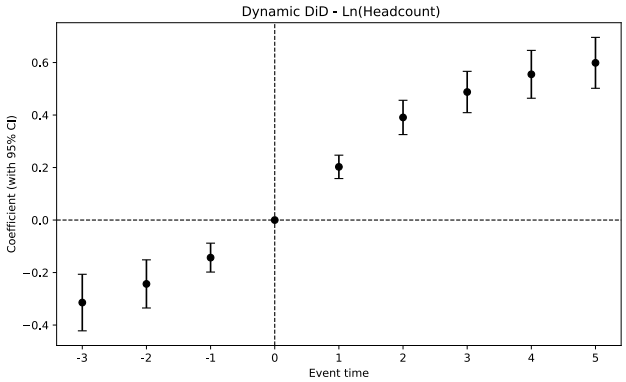
Figure 6: Difference-in-Differences Estimates for Technical Debt Measures

3.6 Firm Scale and Engineering Capacity

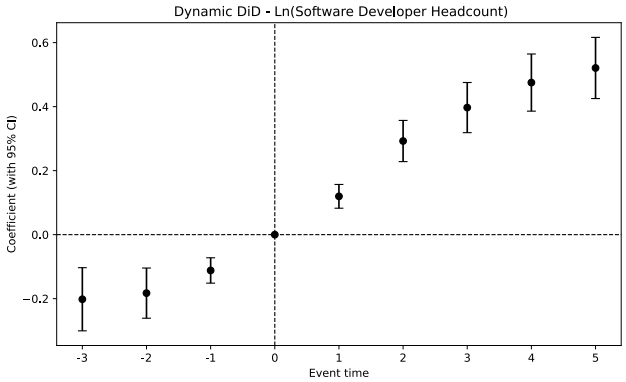
In this section, we examine the dynamics of firm scale and engineering capacity to contextualize the environment in which the previously observed quality improvements occur. Figure 7 plots the dynamic difference-in-differences coefficients for the natural logarithm of lines of code (Panel a), total firm headcount (Panel b), and software engineering headcount (Panel c).



(a) Difference-in-Differences Estimate - Ln(# of Lines of Code)



(b) Difference-in-Differences Estimate - Ln(Total # of Employees)



(c) Difference-in-Differences Estimate - Ln(# of Software Engineers)

Figure 7: Dynamic Difference-in-Differences Estimates for Firm and Product Scaling

We observe a post-investment expansion consistent with a noteworthy resource injection—a sharp and statistically significant discontinuity in organizational scale following the funding event ($t > 0$). Panels (b) and (c) show that VC-backed firms expand their workforce, with the number of software engineers increasing by approximately 45–57 percentage points faster relative to controls by $t = 5$. Panel (a) indicates that while the codebase size grows, the expansion in lines of code is slower than the expansion in headcount. This suggests a slightly lower marginal code output per developer,

consistent with the coordination costs inherent in rapidly scaling engineering teams (i.e., training new junior engineers).

We acknowledge that the parallel trends assumption is violated for these scale measures in the pre-treatment period ($t < 0$). As visible in all three panels, treated firms exhibit a steeper growth trajectory than controls prior to the funding event. This confirms a stylized fact in entrepreneurial finance: venture capitalists select on fast-growing firms. The treated firms are, as we show, different from the average control firm in the cohort regarding their commercial traction and hiring velocity.

However, it is crucial to distinguish between the trajectory of organizational scale and the trajectory of engineering quality. While treated firms diverge in headcount prior to treatment, they do not diverge in bug density or reliability issues until after the investment (as shown in Figure 1). If our main results were driven purely by an unobserved "firm quality" factor, such as superior management capability, we would expect to see parallel improvements in both scale and code quality occurring simultaneously. The fact that scaling precedes the investment, while quality improvements are distinct to the post-investment period, supports the interpretation that the VC intervention imposes a specific governance discipline that is not merely a proxy for the firm's existing growth momentum.

To address the concern that these different growth rates might correlate with unobserved engineering sophistication or management quality shifts, we explicitly match on pre-treatment headcount growth in our robustness checks (Section 4). By constructing a control group that mimics the scaling trajectory of the treated firms, we aim to further isolate the effect of the VC investment from the effect of organizational expansion.

Finally, an alternative interpretation is that post-funding improvements in engineering quality reflect changes in team composition—specifically, that VCs improve code quality by injecting senior technical talent who write better code. We test this directly using LinkedIn employment data. Table 12 reports triple-interaction specifications that interact the treatment effect with two hiring-composition measures: whether the firm hired a technical leader (CTO, VP of Engineering, or equivalent) within six months of funding, and whether the ratio of senior hires exceeded the treated-firm median. Across all eight interaction terms (two hiring dimensions $\times$ four outcomes), none approaches conventional significance, and the base treatment effect remains significant in every specification. Combined with the angel-investor null reported in Table 11—where capital without governance produces no quality gains—this evidence supports a governance-and-process mechanism, which we discuss in detail in Section 3.9.

3.7 Defect Levels vs. Defect Density

The preceding results establish that VC-backed firms experience significant reductions in defect *density*—critical violations per 1,000 lines of code, total violations per 1,000 lines of code, and technical debt per line of code—while simultaneously expanding their codebases by approximately

14.6% (exp(0.136) − 1). A natural question is whether the density improvements reflect a genuine change in engineering practices or are simply a mechanical consequence of codebase growth diluting a fixed stock of defects. To distinguish these interpretations, we estimate our baseline specification using *level* outcomes—the natural logarithm of total violations, the natural logarithm of critical violations, and the natural logarithm of technical debt—which are unaffected by denominator growth.

Table 8 presents the results. Columns (1)–(2) report the per-LoC (density) coefficients from our main tables for ease of comparison. Columns (3)–(4) report the corresponding levels regressions using the log of absolute counts. The contrast is informative. While critical violations per 1,000 lines of code decline by 2.92 (Column 1, significant at the 1% level) and technical debt per line of code declines by $0.97 (Column 2, significant at the 5% level), the total *number* of critical violations ($\hat{\beta}$ = 0.019, SE = 0.088) and the total dollar value of technical debt ($\hat{\beta}$ = 0.104, SE = 0.083) remain statistically indistinguishable from their pre-treatment levels. Total violations across all severity levels are similarly unchanged ($\hat{\beta}$ = 0.071, SE = 0.075; not tabulated).

Table 8: Density vs. Level Outcomes: Distinguishing Practice Improvement from Dilution

	Density		**Levels (log counts)**	
	Crit. Violations / kLoC	TD per LoC ($)	Ln(Crit. Violations)	Ln(TD $)
	(1)	(2)	(3)	(4)
Treated × Post	-2.923***	-0.965**	0.019	0.104
	(1.094)	(0.460)	(0.088)	(0.083)
Cohort × Firm FE	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes
# Obs	325,029	325,029	325,029	325,029
R^2	0.820	0.777	0.921	0.924

This table contrasts the treatment effect on defect *density* (per unit of code) versus defect *levels* (log counts). Columns (1)–(2) reproduce the baseline per-LoC results from Tables 3 and 7. Columns (3)–(4) estimate the same specification using the natural logarithm of the absolute count of critical violations and the natural logarithm of total technical debt (in dollars). All specifications include Firm × Cohort and Month × Cohort fixed effects. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses, clustered by cohort and firm. * p<0.1, ** p<0.05, *** p<0.01.

This pattern—significant density improvements coupled with stable absolute counts—is precisely what we should expect if the mechanism is *improved engineering practices on new code* rather than retroactive remediation of legacy defects. Software defects are an inherent byproduct of the coding process: each line of code carries a probability of introducing a violation. The defect *rate*—violations per unit of code—is therefore the appropriate metric for the quality of engineering practices, analogous to how a manufacturing defect rate (defects per unit produced) captures process quality rather than cumulative output. Post-investment, treated firms write approximately 14.6% more code without

proportionally increasing their defect counts, implying that new code is produced to a measurably higher standard.

We note that the point estimates for the levels outcomes, while statistically insignificant, are weakly positive (0.019 for critical violations, 0.071 for total violations, 0.104 for technical debt). These estimates are consistent with a modest increase in absolute defect counts as the codebase grows—but an increase far smaller than the 14.6% expansion in code volume. If defect rates had remained constant, we would expect log defect counts to increase by approximately 0.136 (matching the log LoC growth). The observed increases of 0.019–0.104 fall well below this benchmark, confirming that defects accumulate at a substantially slower rate than code is produced. Full event-study figures for the levels outcomes are presented in Appendix G.

3.8 Decomposing the Quality Improvement

While the levels decomposition in the preceding subsection provides initial evidence against a pure dilution interpretation, we subject this question to two additional, more stringent tests. Together, these three analyses establish that the decline in defect density reflects genuine quality improvement, not merely an arithmetic consequence of codebase expansion.

Arithmetic benchmark.

We first note that the magnitudes of the density improvements substantially exceed what mechanical dilution can explain. Codebase size increases by approximately 14.6% ($\exp(0.136) - 1$), yet critical violations per 1,000 lines of code decline by 2.92 against a treated-firm baseline mean of 12.36—a 24% reduction. Even under the extreme (and counterfactual) assumption that not a single new defect accompanies the additional code, a 14.6% increase in the denominator can produce at most a $1 - (1/1.146) = 12.7\%$ decline in defect density. The observed 24% reduction is nearly twice this upper bound, indicating that at least half of the improvement must originate from a genuine change in defect generation rates.

Stable-LoC subsample.

To test this directly, we restrict the sample to firm-cohort combinations in the bottom quartile of post-treatment codebase growth. Among these firms—whose codebases are essentially stable—the denominator of the per-LoC ratio changes minimally, and any observed density improvement cannot be attributed to dilution.

Table 9 reports the results. Column (1) reproduces the full-sample baseline for reference. Column (2) estimates the identical specification on the stable-LoC subsample. For critical violations per 1,000 lines of code, the treatment effect in the stable-LoC subsample is −4.534 (SE = 1.939,

$p < 0.05$)—55% *larger* in absolute value than the full-sample estimate of −2.923. For technical debt per line of code, the stable-LoC coefficient is −1.000 (Column 4, SE = 0.485, $p < 0.05$), essentially identical to the full-sample estimate of −0.965 (Column 3). In both cases, the effects survive—and in the case of critical violations, amplify—when the denominator channel is neutralized.

Table 9: Stable-LoC Subsample: Defect Density Improvements Without Codebase Growth

	Critical Violations / kLoC		**TD per LoC**	
	Full Sample	Stable LoC (Q1)	Full Sample	Stable LoC (Q1)
	(1)	(2)	(3)	(4)
Treated × Post	-2.923***	-4.534**	-0.965**	-1.000**
	(1.094)	(1.939)	(0.460)	(0.485)
Cohort × Firm FE	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes
# Obs	325,029	318,217	325,029	318,217
R^2	0.820	0.820	0.777	0.777

This table compares the baseline per-LoC estimates (Columns 1 and 3) with estimates from a subsample restricted to firm-cohort observations in the bottom quartile of post-treatment codebase growth (Columns 2 and 4). The stable-LoC restriction removes treated firms whose codebases expanded most aggressively, isolating the quality effect from denominator-driven dilution. All specifications include Firm × Cohort and Month × Cohort fixed effects. See Appendix A, Table 16, for variable definitions. *Note:* Standard errors in parentheses, clustered by cohort and firm. * p<0.1, ** p<0.05, *** p<0.01.

The amplification of the critical violations result in the stable-LoC subsample merits comment. Firms that do not substantially expand their codebases after VC investment may allocate more of their engineering effort toward remediation and process improvement rather than new feature development, producing a larger per-LoC reduction. Alternatively, the full-sample estimate may be attenuated by a compositional effect: firms with the fastest codebase growth may add proportionally more violations on their marginal code, partially offsetting the practice improvement. Regardless of the precise interpretation, the stable-LoC result decisively rules out the hypothesis that our main findings are a pure artifact of denominator growth.

Poisson pseudo-maximum likelihood.

As a complementary parametric test, we estimate a Poisson pseudo-maximum likelihood (PML) model for critical violation *counts*, explicitly controlling for codebase size via log(LoC). This approach models defects as counts and directly conditions on the denominator, avoiding the ratio construction altogether.

Table 10: Poisson PML Estimates for Critical Violation Counts

Dep Var:	**Critical Violations**	**Violations (All)**
	(1)	(2)
Treated × Post	-0.062*	-0.056
	(0.035)	(0.039)
log(LoC)	0.943***	0.992***
	(0.072)	(0.107)
Cohort × Firm FE	Yes	Yes
Month × Cohort FE	Yes	Yes
# Obs	307,903	325,029

This table reports Poisson pseudo-maximum likelihood estimates with log(LoC) as a direct control for codebase size. The dependent variables are the raw counts of critical violations and total violations (not normalized per LoC). Cohort × Firm fixed effects are included. The coefficient on Treated × Post is interpretable as a semi-elasticity: $\exp(-0.062) - 1 = -6.0\%$. The observation count in Column (1) is lower because Poisson estimation with fixed effects drops firm-cohort groups with zero variation in the outcome. *Note:* Standard errors in parentheses, clustered by cohort and firm. * p<0.1, ** p<0.05, *** p<0.01.

Column (1) of Table 10 reports a treatment coefficient of -0.062 (SE = 0.035), significant at the 10% level. This implies that treated firms experience an $\exp(-0.062) - 1 = -6.0\%$ reduction in critical violations, holding codebase size constant. The coefficient on log(LoC) is 0.943—significantly below 1.0—confirming that violations scale *less than proportionally* with code volume. A 10% increase in codebase size is associated with only a 9.4% increase in critical violations, consistent with larger codebases benefiting from shared infrastructure and reusable components that reduce per-unit defect rates. Column (2) reports a negative but statistically insignificant treatment effect for total violations ($\hat{\beta} = -0.056$, SE = 0.039), consistent with the weaker per-LoC results for this broader measure.

Taken together, the three tests converge on the same conclusion. The arithmetic bound establishes that dilution alone cannot explain the magnitude of the observed improvement. The stable-LoC subsample shows that the effect survives—and amplifies—when codebase growth is neutralized. The Poisson PML confirms a 6.0% reduction in critical violations after directly conditioning on code volume. We interpret these results as strong evidence that the decline in defect density following VC investment reflects a genuine improvement in engineering practices rather than a mechanical denominator effect.

3.9 Capital vs. Monitoring: Mechanism Evidence

The results thus far establish that first-time VC investment is associated with improved engineering practices, as measured by lower defect densities per unit of code. A natural question is whether this improvement is driven by the *capital* that accompanies VC investment or by the *monitoring and governance* that institutional investors impose. The distinction is economically important: if capital alone suffices to improve engineering quality—through hiring senior engineers, purchasing better tooling, or funding technical debt remediation—then any source of funding should produce similar effects. If, instead, the quality improvement requires the governance infrastructure that institutional VCs provide—board oversight, operating partners with engineering expertise, portfolio-wide best practices—then capital without monitoring should leave defect densities unchanged.

To test this, we exploit variation in the structure of the first VC deal. When a financing round involves multiple co-investors, monitoring incentives are more dispersed: each investor bears the private costs of oversight while sharing the benefits of improved firm performance across all syndicate members (Lerner, 1994; Casamatta and Haritchabalet, 2007; Kang, Li, and Oh, 2022). If monitoring is the operative channel, firms financed by larger syndicates should experience smaller quality improvements than those with more concentrated ownership. We classify a treated firm as *syndicated* if the first financing deal involves more than the sample median number of co-investors (five). This yields 103 syndicated and 111 non-syndicated treated firms, providing a balanced split with adequate statistical power. We interact the treatment indicator with this syndicate classification and estimate:

$$M_{i,c,t} = \beta_1 VC_{i,c,t} + \beta_2 VC_{i,c,t} \times \text{Syndicate}_i + \beta_3 \text{Post}_{c,t} \times \text{Syndicate}_i + \chi_{c,i} + \delta_{c,t} + \varepsilon_{i,c,t} \tag{7}$$

where β_1 captures the treatment effect for firms with fewer co-investors and $\beta_1 + \beta_2$ gives the total effect for syndicated firms. If monitoring drives the quality improvement, we expect $\beta_2 > 0$: firms with more co-investors should experience smaller quality gains.

Table 11 reports the results. Columns (3), (6), and (9) present the syndicate specifications. For critical violations per 1,000 lines of code, Column (3) reports a base treatment effect of −3.631 (SE = 1.377, $p < 0.01$) among non-syndicated firms. The syndicate interaction is +1.159 (SE = 2.345), yielding a net effect for syndicated firms of −3.631 + 1.159 = −2.472, smaller in magnitude and no longer statistically significant. The pattern is similar for technical debt per line of code: Column (6) reports a base effect of −1.417 (SE = 0.815, $p < 0.10$) with a syndicate interaction of +0.838 (SE = 0.948), producing a net syndicated effect of −0.579. In both cases, the interaction terms are positive and consistently attenuate the base effect, suggesting that firms financed by fewer co-investors experience larger quality improvements than those in larger syndicates, consistent with a monitoring

rather than capital mechanism. Columns (1), (4), and (7) reproduce the baseline treatment effects for reference.

Table 11: Capital vs. Monitoring: Angel and Syndicate Heterogeneity

	Critical Violations / kLoC			**TD per LoC ($)**			**Ln(Lines of Code)**		
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Treated × Post	-2.923***	-2.984***	-3.631***	-0.965**	-0.995**	-1.417*	0.136*	0.134*	0.079
	(1.094)	(1.127)	(1.377)	(0.460)	(0.474)	(0.815)	(0.072)	(0.072)	(0.089)
Post × Angel		-0.692			-0.097			-0.073	
		(1.223)			(0.195)			(0.046)	
Treated × Post × Angel		2.525*			1.093**			0.110	
		(1.517)			(0.536)			(0.231)	
Post × Syndicate			0.349			0.088			0.073
			(0.691)			(0.298)			(0.053)
Treated × Post × Syndicate			1.159			0.838			0.062
			(2.345)			(0.948)			(0.135)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	325,029	325,029	325,029	325,029	325,029	325,029	325,029	325,029	325,029
R^2	0.820	0.820	0.820	0.777	0.777	0.777	0.901	0.901	0.901

This table reports heterogeneity in the treatment effect by investor type. Syndicate is an indicator equal to one if the first VC deal involves more than the sample median number of co-investors (five). Angel is an indicator equal to one if the firm received angel financing at any point in its PitchBook deal history (9 of 225 treated firms). Columns (1), (4), and (7) reproduce the baseline estimates. Columns (2), (5), and (8) introduce the angel interaction; the net treatment effect for angel-backed firms is $\hat{\beta}_1 + \hat{\beta}_2$. Columns (3), (6), and (9) introduce the syndicate interaction. All specifications include Firm × Cohort and Month × Cohort fixed effects. See Appendix A, Table 16, for variable definitions. *Note:* Standard errors in parentheses, clustered by cohort and firm. * p<0.1, ** p<0.05, *** p<0.01.

As corroborating evidence, Columns (2), (5), and (8) introduce an angel interaction. We classify a firm as *angel-backed* if it received angel financing at any point in its PitchBook deal history. Unlike institutional VCs, who receive board seats and contractual control rights as standard governance tools (Kaplan and Strömberg, 2003; Lerner, 1995), angel investors typically do not impose formal governance structures on portfolio companies (Ibrahim, 2008; Chemmanur and Chen, 2014). Only 9 of the 225 treated firms are angel-backed, so this test has limited statistical power and we interpret it as directional evidence. The results are nonetheless suggestive: for critical violations, the base treatment effect among institutionally-backed firms is -2.984 (SE = 1.127, $p < 0.01$), while the angel interaction is +2.525 (SE = 1.517, $p < 0.10$), yielding a net effect for angel-backed firms of -0.459—economically negligible. For technical debt, Column (5) reports a similar pattern: the angel interaction is +1.093 (SE = 0.536, $p < 0.05$), fully offsetting the base effect. Angel-backed firms experience no measurable improvement in engineering quality following investment, yet they scale

their codebases at comparable rates (Column 8: angel interaction on log(LoC) is +0.110, SE = 0.231, statistically insignificant). The differentiation is entirely on the quality dimension, not the scaling dimension, consistent with capital alone being insufficient to improve engineering practices.

In a complementary exercise, we estimate a separate stacked DiD where the treatment is receipt of angel financing. Across all four outcomes, the angel-as-treatment coefficients are insignificant and carry the opposite sign to the baseline, consistent with capital alone being insufficient. However, with only five angel-first firms, this test has limited power; full results appear in the Online Appendix.

Human Capital Injection vs. Governance

The investor-type tests above suggest that the quality improvement requires institutional governance. A complementary alternative explanation is that the quality improvement operates through a *human capital injection* channel: VCs improve code quality not through governance per se, but by facilitating the hiring of senior technical talent who write better code. To distinguish this channel from a governance mechanism, we interact the treatment indicator with two measures of post-funding hiring composition constructed from LinkedIn employment data. The first, *Hired Technical Leader*, is an indicator equal to one if the firm hired a CTO, VP of Engineering, or equivalent technical executive within six months of the funding event. The second, *High Senior Ratio*, is an indicator equal to one if the ratio of senior engineers among all new engineering hires exceeds the treated-firm median. We then estimate:

$$M_{i,c,t} = \beta_1 VC_{i,c,t} + \beta_2 VC_{i,c,t} \times H_i + \beta_3 \text{Post}_{c,t} \times H_i + \chi_{c,i} + \delta_{c,t} + \varepsilon_{i,c,t} \tag{8}$$

where H_i denotes the hiring-composition indicator. If the quality improvement reflects human capital injection, we expect $\beta_2 < 0$: firms that hire senior technical leaders should experience larger improvements. If, instead, governance drives the effect, β_2 should be close to zero—the improvement should manifest regardless of team composition changes.

Table 12 reports the results for four engineering quality outcomes. Across all eight interaction terms (two hiring dimensions × four outcomes), none approaches conventional significance. For critical violations per 1,000 lines of code, the hired-technical-leader interaction is +1.261 (SE = 2.376) and the high-senior-ratio interaction is −2.256 (SE = 2.850). The pattern is similar for total violations (+4.626, SE = 5.419; −2.376, SE = 5.710), technical debt per line of code (+0.911, SE = 0.879; +0.096, SE = 0.903), and maintainability issues (+3.845, SE = 4.980; −1.716, SE = 5.122). Importantly, the base treatment effect (β_1) remains significant in every specification, indicating that introducing the hiring interactions does not absorb the treatment effect. The quality improvement is orthogonal to observable changes in team composition.

Table 12: Human Capital vs. Governance: Hiring-Composition Heterogeneity

	Critical Violations / kLoC			Violations / kLoC			TD per LoC ($)			Maint Issues / kLoC		
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
Treated × Post	-2.923***	-3.630**	-2.314**	-6.897**	-8.959**	-6.579*	-0.965**	-1.406*	-1.086	-6.461**	-8.381**	-6.554*
	(1.094)	(1.801)	(1.091)	(2.795)	(4.088)	(3.692)	(0.460)	(0.733)	(0.684)	(2.621)	(3.979)	(3.652)
Post × Hired Technical Leader		0.378			-0.027			0.086			0.525	
		(0.941)			(1.709)			(0.323)			(1.659)	
Treated × Post × Hired Technical Leader		1.261			4.626			0.911			3.845	
		(2.376)			(5.419)			(0.879)			(4.980)	
Post × High Senior Ratio			0.947			1.922			0.229			2.305
			(0.737)			(1.486)			(0.296)			(1.407)
Treated × Post × High Senior Ratio			-2.256			-2.376			0.096			-1.716
			(2.850)			(5.710)			(0.903)			(5.122)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	325,029	325,029	325,029	325,029	325,029	325,029	325,029	325,029	325,029	325,029	325,029	325,029
R^2	0.820	0.820	0.820	0.817	0.817	0.817	0.777	0.777	0.777	0.790	0.790	0.790

This table reports heterogeneity in the treatment effect by post-funding hiring composition. Hired Technical Leader is an indicator equal to one if the firm hired a CTO, VP of Engineering, or equivalent technical executive within six months of the funding event. High Senior Ratio is an indicator equal to one if the ratio of senior engineers among all new engineering hires exceeds the treated-firm median. Columns (1), (4), (7), and (10) reproduce the baseline estimates. Columns (2), (5), (8), and (11) introduce the hired-technical-leader interaction. Columns (3), (6), (9), and (12) introduce the high-senior-ratio interaction. All specifications include Firm × Cohort and Month × Cohort fixed effects. See Appendix A, Table 16, for variable definitions. *Note:* Standard errors in parentheses, clustered by cohort and firm. * p<0.1, ** p<0.05, *** p<0.01.

Synthesis: Governance over Capital and Personnel

We interpret the combined evidence from the investor-type and hiring-composition tests as pointing toward a governance-and-process mechanism. Three alternative or supplementary channels have been tested. First, the syndicate heterogeneity (Table 11, Columns 3, 6, 9) shows that firms financed by fewer co-investors experience larger quality improvements than those in larger syndicates, consistent with a monitoring rather than capital mechanism. Second, the angel null (Table 11, Columns 2, 5, 8) demonstrates that capital without active monitoring is insufficient: the few angel-backed firms in our sample receive funding and scale at comparable rates but exhibit no improvement in defect density. Third, the hiring null (Table 12) demonstrates that the quality improvement does not operate through the observable human-capital-injection channel of recruiting senior technical talent: firms that hire technical leaders experience the same quality improvement as those that do not. Together, these results suggest that the operative mechanism is the governance infrastructure that accompanies institutional VC investment—board oversight, operating partners with engineering expertise, and portfolio-wide best practices—rather than either capital per se or the specific individuals hired with that capital. This interpretation is consistent with a broader literature on the operational value-add of VC investors (Gorman and Sahlman, 1989; Hellmann and Puri, 2002; Bernstein, Giroud, and Townsend, 2016), extended here to the domain of software engineering practices.

We acknowledge several caveats. First, the syndicate indicator is a coarse proxy for governance intensity: deal size, investor reputation, and board seat allocation likely matter more than the raw

count of co-investors, but these finer measures are not uniformly available in our data. Second, the angel subsample is small (9 of 225 treated firms), limiting the power of the angel interaction test; the estimates are directionally consistent with our interpretation but should be viewed as suggestive. Third, the hiring-composition measures are inherently limited: we observe job titles and seniority from LinkedIn profiles, but not the specific technical skills, code review practices, or mentorship activities that these hires bring. A senior hire who quietly establishes code review norms would appear identical to one who has no engineering impact. Moreover, LinkedIn coverage is incomplete—not all treated firms have matched hiring records—and firms without matches are coded as zero (no technical leader hired, low senior ratio), which may attenuate the interaction estimates toward zero. Fourth, we cannot rule out that angel-backed firms differ from institutionally-backed firms on unobservable dimensions—such as founding team ambition, product complexity, or market segment—that independently correlate with engineering quality trajectories. Despite these limitations, the convergence of our tests provides what is, to our knowledge, the first evidence that VC governance, rather than VC capital or VC-facilitated hiring, is associated with measurable improvements in technical practices.

4 Robustness

4.1 Matching on pre-treatment growth

To address the concern that our results are driven by the endogenous selection of high-growth firms, we re-estimate our baseline model using a matching approach. As shown in Figure 7, treated firms exhibit a steeper pre-treatment growth trajectory—in terms of headcount and software assets—relative to control firms. This raises the possibility that our main results capture unobserved firm characteristics associated with rapid scaling rather than the specific governance effect of the venture capitalist.

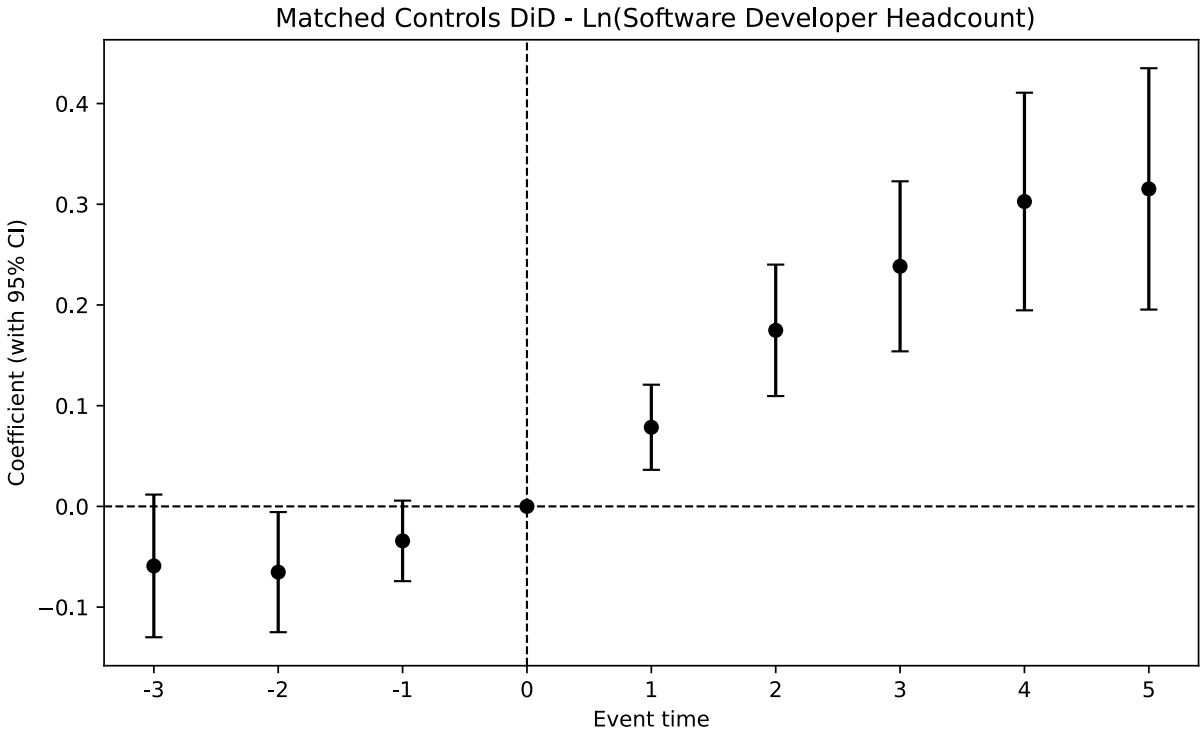
To mitigate this concern, we refine the control group within each cohort based on pre-treatment growth dynamics. Specifically, for each cohort we match treated firms to control firms that exhibit a comparable headcount growth rate in the months leading up to the event ($t < 0$). This yields a comparison between VC-funded firms and other fast-growing startups that followed a similar trajectory but did not receive venture funding at that point in time. We define pre-treatment growth as the difference in log average headcount between $t \in [-4, -2]$ and $t \in [-12, -11]$:

$$\text{Growth}_{i,c}^{\text{pre}} = \ln\left(\frac{1}{3}\sum_{t=-4}^{-2}\text{Headcount}_{i,c,t}\right) - \ln\left(\frac{1}{2}\sum_{t=-12}^{-11}\text{Headcount}_{i,c,t}\right).$$

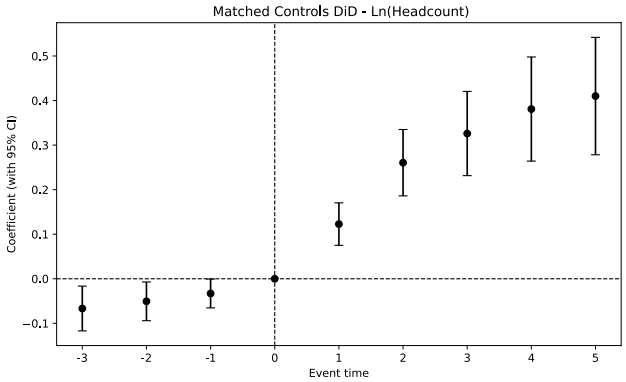
Details on the matching approach can be found in Appendix D.

Figure 8 presents the dynamic evolution of firm scaling outcomes in the matched sample, emphasizing software engineering headcount (Panel a) since our matching criterion is based on

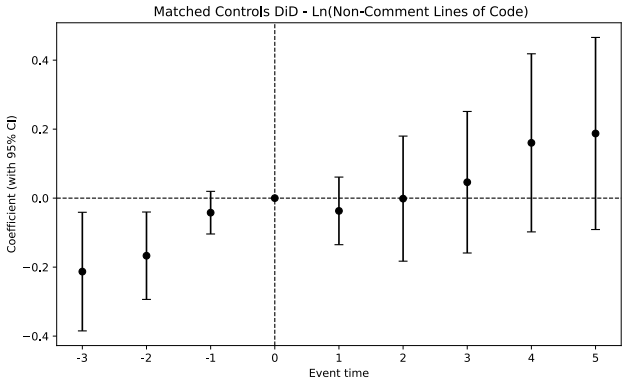
pre-treatment headcount growth. In contrast to the baseline sample (Figure 7), where treated firms exhibited steeper pre-treatment growth trajectories, the matched sample shows parallel pre-trends across all three scaling measures. Panels (a), (b), and (c) demonstrate that treated and control firms follow similar trajectories for software engineering headcount, total headcount, and lines of code during the pre-treatment period ($q < 0$). The coefficients cluster around zero and show no systematic divergence prior to the funding event. This confirms that the matching procedure successfully isolates control firms with comparable growth dynamics, addressing the concern that differential pre-treatment scaling might confound our baseline estimates. Post-treatment, we observe the expected expansion in organizational scale, with treated firms increasing headcount relative to these matched controls.



(a) Difference-in-Differences Estimate - Ln(# of Software Engineers)



(b) Difference-in-Differences Estimate - Ln(Total # of Employees)



(c) Difference-in-Differences Estimate - Ln(# of Lines of Code)

Figure 8: Dynamic Difference-in-Differences Estimates for Firm and Product Scaling (Matched Sample)

Table 13 presents the static difference-in-differences estimates for firm scaling outcomes. Treated firms exhibit a significant increase in total headcount (0.33 log points, $p < 0.01$) and software engineering headcount (0.26 log points, $p < 0.01$) following VC investment, even when compared to control firms with similar pre-treatment growth trajectories. This is consistent with the view that the observed improvements in engineering quality are unlikely to be explained by unobserved firm characteristics correlated with faster pre-funding growth.

Table 13: Difference-in-Differences Estimates for Firm Scaling (Matched Sample)

	Ln(Total Headcount)			**Ln(SWE Headcount)**		
	(1)	(2)	(3)	(4)	(5)	(6)
Treated × Post	0.331***	0.366***	0.352***	0.255***	0.296***	0.212***
	(0.048)	(0.069)	(0.063)	(0.043)	(0.062)	(0.055)
Post × High Violations (=1)		-0.022			-0.001	
		(0.027)			(0.023)	
Treated × Post × High Violations (=1)		-0.076			-0.088	
		(0.088)			(0.075)	
Post × Large Round			0.137***			0.087*
			(0.047)			(0.047)
Treated × Post × Large Round			-0.149			0.041
			(0.103)			(0.088)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	43,968	43,968	43,968	43,968	43,968	43,968
R^2	0.995	0.995	0.995	0.993	0.993	0.993

This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Treated × Post indicator. The sample is restricted to treated firms and control firms matched on pre-treatment headcount growth. See Appendix D for details on the matching procedure. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

Turning to our core engineering quality metrics, Table 14 reports the matched estimates for critical violations, total violations, and maintainability issues. The matching procedure primarily trims the control group by restricting attention to control firms with comparable pre-treatment growth dynamics; the matched stacked sample contains approximately 46,000 observations compared to 325,000 in the baseline—a reduction of roughly 86%. Despite this substantial reduction in sample size, the point estimates remain broadly stable. For critical violations, the matched coefficient (−2.62) is 90% of the baseline estimate (−2.92), while the standard error increases by 27%, consistent with reduced precision in the smaller matched sample. For total violations and maintainability issues, the matched estimates (−9.27 and −8.67, respectively) are larger in magnitude than the baseline (−6.90 and −6.46), though the standard errors increase by 34–44%. Overall, the engineering-quality results are similar in magnitude when comparing treated firms to fast-growing controls.

Table 14: Difference-in-Differences Estimates for Engineering Quality (Matched Sample)

	Critical Violations			**Violations (All)**			**Maintainability Issues**		
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Treated × Post	-2.621*	-0.457	-3.316	-9.267**	-5.859*	-10.815*	-8.666**	-5.374	-9.408*
	(1.390)	(0.546)	(2.178)	(4.015)	(3.444)	(5.630)	(3.515)	(3.286)	(4.822)
Post × High Violations (=1)		-1.883***			-5.530			-4.454	
		(0.709)			(3.604)			(3.335)	
Treated × Post × High Violations (=1)		-4.828*			-7.910			-7.533	
		(2.546)			(7.218)			(6.247)	
Post × Large Round			1.554			-1.750			-0.053
			(2.017)			(3.629)			(3.316)
Treated × Post × Large Round			0.513			4.974			1.807
			(2.921)			(7.342)			(6.583)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	46,243	46,243	46,243	46,243	46,243	46,243	46,243	46,243	46,243
R^2	0.844	0.846	0.845	0.822	0.822	0.822	0.783	0.784	0.783

This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Treated × Post indicator. The sample is restricted to treated firms and control firms matched on pre-treatment headcount growth. Critical Violations, Violations (All), and Maintainability Issues are measured per 1,000 lines of code. See Appendix D for details on the matching procedure and Appendix A, Table 16, for variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

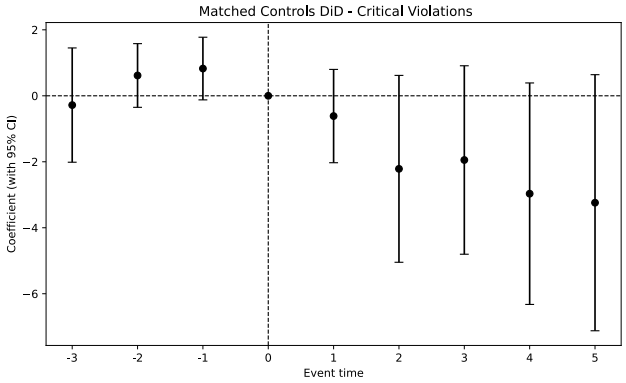
Table 15 presents the matched estimates for our technical debt measures. The matched coefficient for technical debt per line of code (−1.23) is 128% of the baseline estimate (−0.97), while the technical debt ratio coefficient (−0.127) is 128% of the baseline (−0.099). In both cases, standard errors increase by approximately 48%, consistent with reduced precision in the smaller matched sample. The point estimates remain economically meaningful: a $1.23 reduction in technical debt per line of code and a 12.7 percentage point decline in the technical debt ratio are substantial relative to the 20% benchmark used for calibration. Taken together, these comparisons are consistent with the view that the baseline findings are not solely an artifact of comparing treated firms to slower-growing controls.

Table 15: Difference-in-Differences Estimates for Technical Debt (Matched Sample)

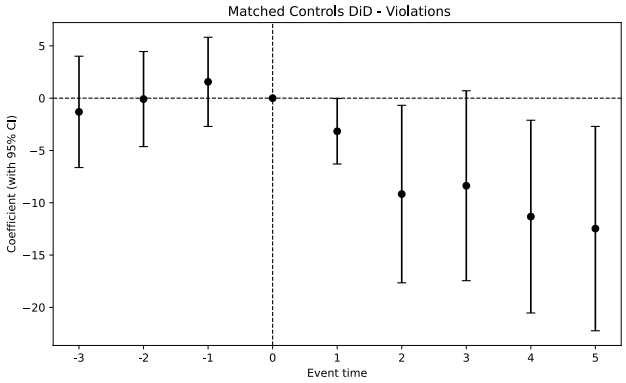
	Technical Debt per LoC			**Technical Debt Ratio**		
	(1)	(2)	(3)	(4)	(5)	(6)
Treated × Post	-1.232*	-0.678	-1.411	-0.127*	-0.067	-0.143
	(0.681)	(0.684)	(0.866)	(0.068)	(0.068)	(0.087)
Post × High Violations (=1)		-0.932			-0.089	
		(0.780)			(0.078)	
Treated × Post × High Violations (=1)		-1.291			-0.138	
		(1.147)			(0.115)	
Post × Large Round			-0.211			-0.020
			(0.735)			(0.073)
Treated × Post × Large Round			0.581			0.053
			(1.306)			(0.131)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	46,243	46,243	46,243	46,243	46,243	46,243
R^2	0.796	0.796	0.796	0.795	0.796	0.795

This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Treated × Post indicator. The sample is restricted to treated firms and control firms matched on pre-treatment headcount growth. Technical Debt per Line of Code and the Technical Debt Ratio are defined in Section 2.5. See Appendix D for details on the matching procedure and Appendix A, Table 16, for variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

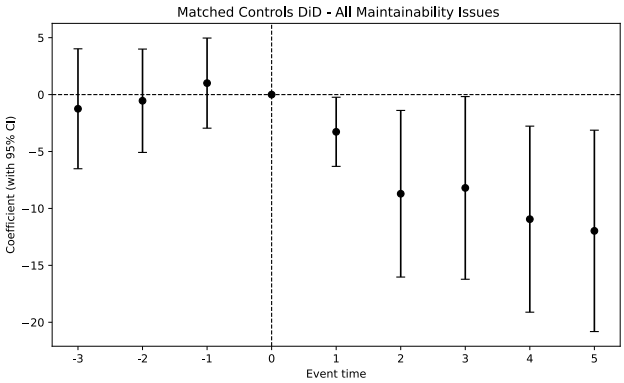
Figures 9 and 10 present the dynamic evolution of the treatment effects in the matched sample. Consistent with the parallel trends assumption, the pre-treatment coefficients are close to zero and statistically insignificant across all specifications. The treatment effects materialize at or shortly after the funding event, corroborating the static estimates presented in the tables above.



(a) Critical Violations per 1,000 LoC

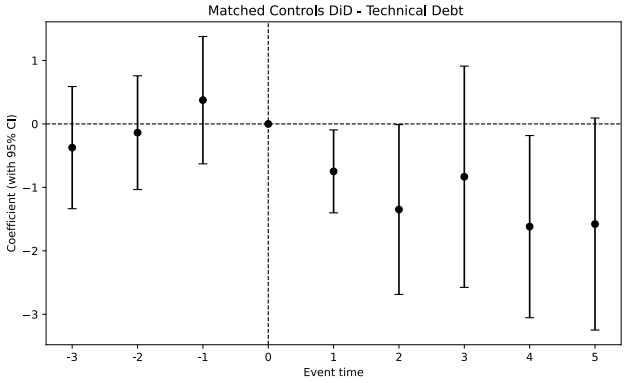


(b) Violations (All) per 1,000 LoC

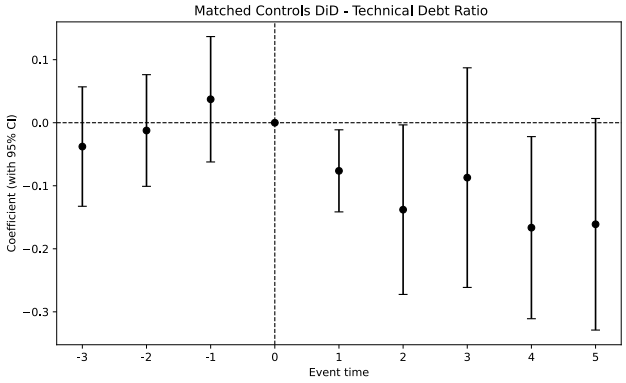


(c) Maintainability Issues per 1,000 LoC

Figure 9: Dynamic Difference-in-Differences Estimates for Engineering Quality (Matched Sample)



(a) Technical Debt per Line of Code



(b) Technical Debt Ratio

Figure 10: Dynamic Difference-in-Differences Estimates for Technical Debt (Matched Sample)

Overall, the matching analysis supports the robustness of our main findings. Across specifications, point estimates remain within 90–134% of their baseline values, while standard errors increase by 27–48%, consistent with reduced precision in the smaller matched sample. Importantly, critical

violations and technical debt per line of code regain marginal significance in the matched sample, while violations and maintainability issues remain significant at 5%. This pattern is consistent with the interpretation that our results are not driven solely by differences in pre-treatment growth trajectories between treated and control firms.

To further assess the sensitivity of the matched estimates to the choice of matching configuration, we conduct a systematic specification-curve analysis (Simonsohn, Simmons, and Nelson, 2020) over 72 configurations that vary the feature set (headcount only; headcount plus lines of code; headcount plus software engineering headcount; all three), the matching mode (centroid, individual $K = 1$, individual $K = 3$), and the caliper ($\delta \in \{0.5, 0.75, 1.0, 1.25, 1.5, 2.0\}$ standard deviations of the within-cohort Mahalanobis distance). Across all 72 specifications, 100% of critical-violation coefficients and 93–99% of total-violation, maintainability, and technical-debt coefficients are negative, with 14–88% reaching significance at the 5% level. The specification curves and full details are reported in Appendix E.

4.2 Stage-Specificity: Later Financing Rounds

A core identification concern is that the engineering-quality improvements we attribute to the first VC entry (Seed/Series A) may instead reflect a more mechanical shift: the relaxation of financial constraints. If the primary driver were simply an infusion of capital that allows firms to hire engineers, refactor code, and invest in process (rather than an initial shift in monitoring and engineering philosophy), then similar discontinuities should appear around later, larger financing events as well. In other words, if "capital" rather than "monitoring/professionalization" explains the baseline results, we would expect measurable improvements in code quality around later funding rounds, when firms receive additional resources and expand engineering capacity in a similar manner.

A complementary hypothesis is that the governance and professionalization norms introduced by venture capitalists—board oversight, engineering standards, disciplined development processes (Hellmann and Puri, 2002; Kaplan and Strömberg, 2003)—are established at initial VC entry and remain in place thereafter. By the time a firm raises a Series B or later round, these monitoring structures are already operational; subsequent financing events inject additional capital and trigger hiring, but they do not replicate the initial installation of oversight infrastructure. This creates a testable asymmetry: if the monitoring and professionalization channel is the primary driver of engineering-quality improvements, we should observe discrete gains at first VC entry but *not* at subsequent rounds, even though the latter are accompanied by comparable resource injections.

We implement a stage-specificity test by re-centering the stacked difference-in-differences design on later-stage financing events. Using PitchBook, we define the event time based on each firm's first Series B-or-later round (Series B, C, D, etc.).

As in the main analysis, we study outcomes in a fixed event-time window of $\kappa_{\text{pre}} = \kappa_{\text{post}} = 12$ months around the cohort month c, where $e = t - c \in [-12, +12]$.

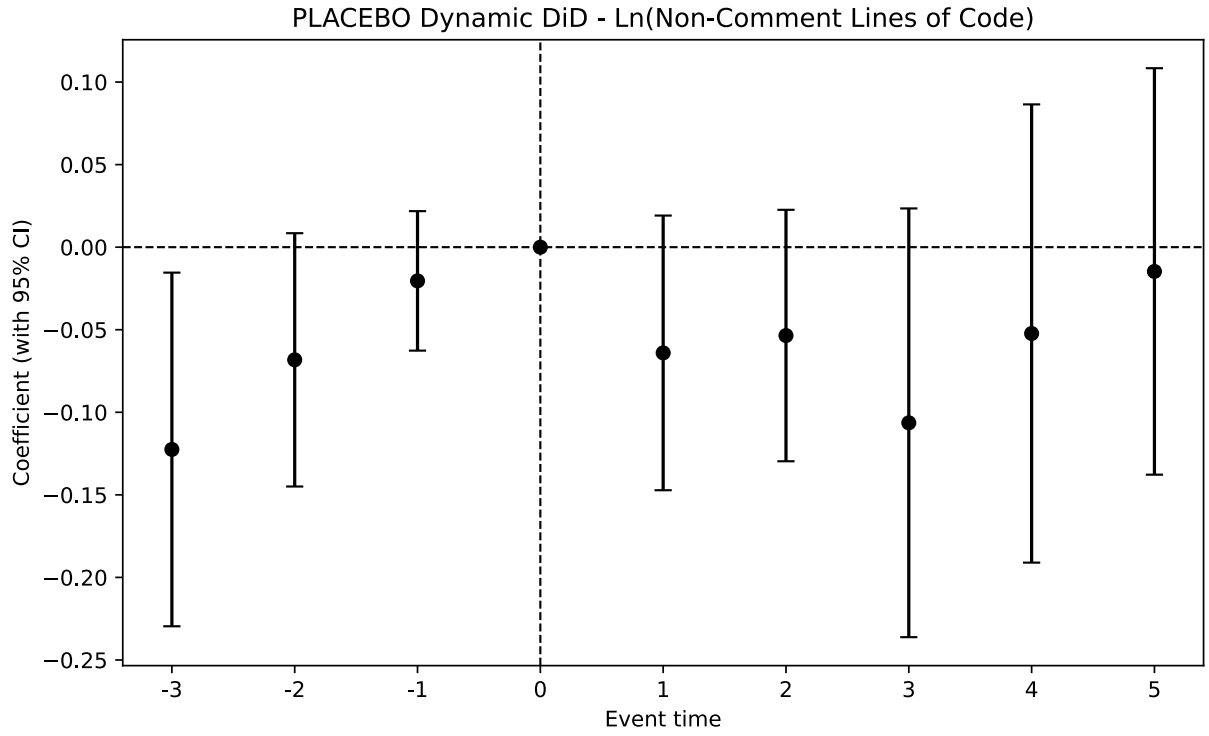
A central nuance in defining a proper control group for this test is avoiding contamination from first VC entry effects during the window. We therefore use clean controls—firms that remain unfunded by VCs throughout each event window. For each Series B+ cohort c, we define clean controls as firms that have not received a Seed/Series A round by the end of the event window. This definition includes firms that never receive VC funding and firms that receive their first VC round only after the event window. This ensures that any discontinuity we estimate at $t = 0$ in the stage-specificity analysis cannot be driven by VC entry among controls.

We estimate the same fixed-effects specification as in the main analysis, replacing the treatment indicator with a Series B+ treatment indicator:

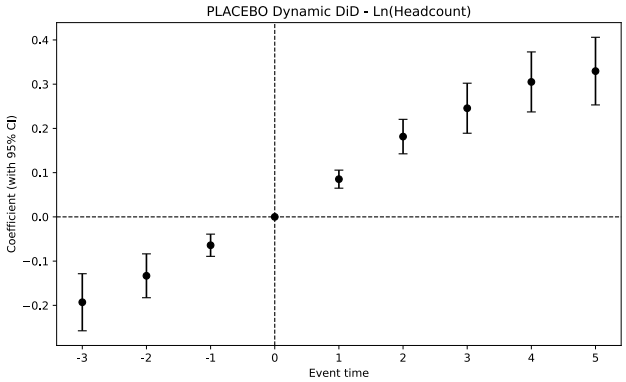
$$M_{i,c,t} = \beta^{B+} VC^{B+}_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}, \tag{9}$$

where B_i denotes firm i's cohort month of its first Series B+ round and $VC^{B+}_{i,c,t} \equiv \mathbf{1}\{B_i = c\} \cdot \mathbf{1}\{t \geq c\}$. As before, $\chi_{c,i}$ and $\delta_{c,t}$ are cohort × firm and cohort × month fixed effects, respectively. Outcomes $M_{i,c,t}$ include the same engineering-quality measures (e.g., critical violations, maintainability issues, technical debt, and related density-normalized metrics) as well as scaling outcomes (total headcount and software engineering headcount). Standard errors are clustered by cohort and firm, consistent with the baseline design.

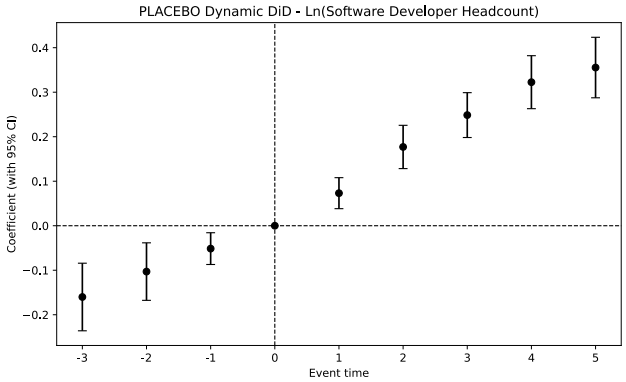
We begin by examining whether Series B+ financing events generate scaling dynamics comparable to those observed around the first VC entry. Figure 11 plots the stage-specificity event-study coefficients for (i) the logarithm of lines of code, (ii) the logarithm of total headcount, and (iii) the logarithm of software engineering headcount, all aligned in event time around the first Series B-or-later round.



(a) (Stage-Specificity) Difference-in-Differences Estimate — Ln(# of Lines of Code)



(b) (Stage-Specificity) Difference-in-Differences — Ln(Total # of Employees)



(c) (Stage-Specificity) Difference-in-Differences — Ln(# of Software Engineers)

Figure 11: Difference-in-Differences Estimates for Firm and Product Scaling (Series B+ rounds)

Two patterns stand out. First, Panels (b) and (c) demonstrate an economically meaningful expansion in organizational scale post-funding. Both total headcount and software engineering headcount increase at $t = 0$ and continue to rise in subsequent months, mirroring the scaling dynamics documented around Seed/Series A in Figure 7 of the main analysis. This confirms that later rounds are also associated with substantial resource injections and rapid hiring, and therefore provide a stringent test of whether improvements in engineering quality can be mechanically attributed to additional capital and staffing.

Second, Panel (a) indicates that product expansion, as proxied by the growth in lines of code, is comparatively slower following later funding rounds. This is consistent with increasing coordination costs and onboarding frictions as engineering teams expand. It is also conservative for our interpretation: slower codebase expansion likely reduces the rate at which new defects are created. Thus, if the engineering-quality improvements documented around first VC entry were driven primarily by increasing engineering capacity (rather than a shift in engineering discipline), the stage-specificity setting should still produce detectable improvements in our engineering-quality and technical debt measures. As we show below, it does not.

The stage-specificity results provide a sharp contrast to our baseline estimates. Around later rounds, we observe similar scaling dynamics: both total headcount and software engineering headcount increase sharply following the financing event, consistent with a resource injection that triggers hiring. Despite this expansion in engineering capacity, we find no corresponding improvement in the engineering-quality outcomes. Across our core software-quality metrics, the Series B+ coefficients are small in magnitude and statistically insignificant, often in the opposite direction of our baseline findings. Furthermore, the dynamic plots show no meaningful discontinuity in engineering quality at the Series B+ event date. All relevant plots and tables can be found in Appendix H.

This exercise strengthens the interpretation that the baseline improvements are not primarily driven by the relaxation of financial constraints. After later rounds, firms still expand, yet they do not generate economically meaningful reductions in technical debt, critical violations, or maintainability issues. The asymmetry is consistent with a monitoring channel: the governance and professionalization standards that drive engineering improvement are established at initial VC entry, not reinstalled at each subsequent capital infusion.

We acknowledge an important caveat: the Series B+ controls include firms that have never received VC funding, which means treated firms in this exercise have already undergone first-VC-entry effects. This asymmetry in prior treatment limits the test's inferential power. Nonetheless, the absence of quality discontinuities at Series B+ rounds—despite comparable scaling dynamics—is informative: it suggests that the engineering-quality improvements documented at first VC entry are specific to the initial installation of governance infrastructure rather than a generic consequence of capital infusion at any stage.

4.3 Additional Robustness Tests

We subject the main findings to a battery of additional specification and design checks. The results are summarized here; full tables and figures appear in the Online Appendix.

Donut event study. One concern is that the treatment effect is concentrated in the months immediately surrounding the funding event, reflecting either anticipation effects (firms cleaning up code in

preparation for due diligence) or measurement timing artifacts (SonarQube scans coinciding with deal closure). We address this by estimating donut specifications that exclude months adjacent to the treatment date. Removing one month on each side of $t = 0$ (donut radius $r = 1$) yields a critical-violations coefficient of -3.339 (SE = 1.264, $p < 0.01$), and extending the exclusion to two months ($r = 2$) produces -3.477 (SE = 1.282, $p < 0.01$). The same monotonic amplification appears for technical debt (-1.085 at $r = 1$, -1.158 at $r = 2$, both significant at the 5% level) and total violations (-7.859 at $r = 1$, -8.377 at $r = 2$, both significant at the 5% level). The fact that effects strengthen—rather than attenuate—when treatment-adjacent months are removed rules out anticipation and measurement timing as explanations for the main results.

Window sensitivity. We re-estimate the baseline specification across five symmetric event windows: ± 6, ± 9, ± 12 (baseline), ± 18, and ± 24 months. Critical violations per 1,000 lines of code remain in the narrow band $[-2.58, -3.03]$ across all five windows, statistically significant at the 5% level or better in every case. Technical debt per line of code ranges from -0.72 to -1.34, and total violations from -4.61 to -8.73, all significant. The coefficient magnitudes are largest at the ± 18 window, consistent with quality improvements that compound as more new code is written to the higher standard.

Leave-one-cohort-out. We systematically re-estimate the baseline specification dropping each of the 85 cohorts in turn. For critical violations, the coefficient ranges from -2.25 to -3.05 (mean $= -2.92$), with the most influential cohort (2015M3) shifting the estimate by 23%. For technical debt, the range is $[-0.64, -1.02]$ (mean $= -0.97$). No single cohort drives any result: all 85 coefficients remain negative for every outcome.

Control reuse in the stacked design. A feature of stacked difference-in-differences designs is that control firms may appear in multiple cohort-specific sub-experiments, inducing correlation across stacked observations. In our sample, the 330 control firms appear in a mean of 40.6 cohorts each (median 35, maximum 85). We address this in two ways. First, we re-weight observations by the inverse of each control firm's appearance frequency, down-weighting heavily reused controls. This specification yields a critical-violations coefficient of -3.425 (SE = 1.280, $p < 0.01$), technical debt of -1.193 (SE = 0.583, $p < 0.05$), and total violations of -8.301 (SE = 3.616, $p < 0.05$)—all magnitudes slightly larger than the unweighted baseline. Second, we impose hard caps on the maximum number of cohorts in which any control firm can appear: at a cap of 50, all three outcomes remain significant at the 5% level or better with magnitudes close to the baseline. More extreme restrictions (caps of 10 and 25) reduce precision substantially but produce no sign flips. Full results are reported in the Online Appendix.

Censoring at next financing. If treated firms receive subsequent financing within the post-treatment window, the post-investment environment may reflect the influence of a later round rather than the initial VC entry. We censor observations after a firm's next financing event. This restriction removes only 34 of 325,029 observations, and the coefficients are unchanged to four decimal places. The test confirms that virtually no firms in our sample receive subsequent financing within the 12-month post-treatment window.

Seed vs. Series A split. We estimate separate regressions for firms whose treatment event is a Seed round (163 firms) and those whose treatment is a Series A round (62 firms). Total violations per 1,000 lines of code is remarkably stable across the two subsamples: -6.930 (SE = 3.520, $p < 0.05$) for Seed and -6.762 (SE = 2.935, $p < 0.05$) for Series A. Critical violations shows a larger point estimate for Series A (-5.493 vs. -1.971), consistent with more intensive monitoring at later stages, though the Series A estimate is only marginally significant given the smaller subsample. No sign flips appear. These results are reported in the Online Appendix.

Language heterogeneity. We conduct leave-one-language-out exercises to verify that no single programming language drives the aggregate results. Across all five languages (Python, Go, JavaScript, Java, TypeScript), removing any one language produces negative coefficients for all three core outcomes. Dropping Go—a language whose built-in compiler and tooling already enforce strict quality standards—strengthens the critical-violations coefficient from -2.923 to -3.831, while dropping Python modestly attenuates it to -2.417. Full language-specific regressions and the leave-one-language-out table appear in the Online Appendix.

5 Conclusion

This paper brings high-frequency software production data to a classic question in entrepreneurial finance: what do venture capitalists change inside firms after they invest? Using longitudinal codebase data on 649 open-core technology companies from 2010–2025, we construct granular measures of engineering quality and develop the first dollar-denominated measure of technical debt in the finance literature. Our stacked difference-in-differences estimates indicate that first-time VC entry is associated with a meaningful reduction in defect density: approximately 2.9 fewer critical violations per 1,000 lines of code (a 24% decline relative to the treated-group mean), 6.5 fewer maintainability issues per 1,000 lines of code, and \$0.97 less technical debt per line of code.

A central question for interpreting these results is whether the per-unit-of-code improvements reflect genuine changes in engineering practices or a mechanical artifact of codebase expansion. Level regressions show that total defect counts remain stable despite substantial codebase growth, consistent with new code being written to a higher standard rather than remediation of legacy issues. We subject

this interpretation to three additional tests. First, the arithmetic magnitude of the defect-rate decline exceeds what the 14.6% increase in codebase size can mechanically explain. Second, restricting the sample to firms with the most stable codebases—where denominator growth is minimal—amplifies the critical-violation effect rather than attenuating it. Third, Poisson pseudo-maximum-likelihood models that explicitly control for code volume confirm a 6.0% reduction in critical violations. Together, these tests establish that the defect-density improvements reflect a genuine shift in engineering practices—what we characterize as scaling discipline: the ability to expand a codebase without proportional quality degradation.

Our heterogeneity analysis points to monitoring and governance as the operative channel. Angel-backed firms—which receive capital but typically lack the governance infrastructure of institutional investors—scale their codebases at comparable rates but exhibit no improvement in defect density. The quality improvements are concentrated among firms backed by institutional VCs with board representation and operating resources. Furthermore, firms that hire senior technical leaders after funding experience the same quality improvement as those that do not—the hiring-composition interactions are uniformly insignificant across all code quality outcomes. This combination of results is difficult to reconcile with either a pure capital story or a human-capital-injection story and instead suggests that the governance apparatus accompanying institutional investment—board oversight, portfolio operating partners, engineering advisors—drives the improvement in development practices. The balance table (Table 2) further supports a causal interpretation: treated and control firms exhibit near-identical baseline defect densities (standardized difference of -0.006 for critical violations per 1,000 lines of code), ruling out a mechanical "room for improvement" selection story. Two additional exercises corroborate the main findings: matching on pre-treatment headcount growth produces similar estimates, and a stage-specificity test around later Series B+ rounds reveals scaling without corresponding quality gains.

We acknowledge several limitations. Our identification strategy exploits within-firm variation around VC entry, but VC financing is not randomly assigned; we cannot fully rule out that unobserved firm trajectories coincide with the timing of investment. The angel-versus-institutional comparison, while informative, is an imperfect proxy for monitoring intensity—some angels provide deep technical governance, and some institutional investors are passive. Although the hiring-composition analysis rules out that the quality improvement is differentially associated with recruiting senior technical leaders, we cannot observe the specific internal process changes—such as the adoption of code review mandates, continuous integration pipelines, or formal testing requirements—that may constitute the governance channel. Our measures capture engineering practices as recorded by static analysis tools, not code quality per se; dimensions such as test coverage, architectural coherence, and developer productivity lie outside our measurement framework. Finally, the open-core sample, while offering unique observability, may not generalize to proprietary-software startups where codebases are not publicly accessible.

These findings contribute to three literatures. First, we extend the VC professionalization narrative from organizational and governance domains (Hellmann and Puri, 2002) to the technical engineering practices through which startups build their core products, documenting that the improvements are concentrated in the most consequential defect categories rather than superficial code issues. Second, we complement the "operational engineering" literature in buyout markets (Bernstein and Sheen, 2016) by showing that a similar dynamic operates at an earlier stage of firm development: the first institutional investor appears to bring not only capital and governance but also a shift toward more disciplined software development. Third, we contribute to the growing literature on intangible assets and software-intensive firms by constructing a dollar-denominated measure of technical debt that makes software liabilities tractable for financial research—a framework that may prove useful for technology-firm valuation, M&A due diligence, and the study of software assets as off-balance-sheet liabilities.

References

Amornsiripanitch, N., P. A. Gompers, and Y. Xuan. 2019. More than money: Venture capitalists on boards. *The Journal of Law, Economics, and Organization* 35:513–43.

Banker, R., Y. Liang, and N. Ramasubbu. 2021. Technical debt and firm performance. *Management Science* 67:3174–94.

Bernstein, S., X. Giroud, and R. R. Townsend. 2016. The impact of venture capital monitoring. *The Journal of Finance* 71:1591–622.

Bernstein, S., and A. Sheen. 2016. The operational consequences of private equity buyouts: Evidence from the restaurant industry. *The Review of Financial Studies* 29:2387–418.

Callaway, B., and P. H. Sant'Anna. 2021. Difference-in-differences with multiple time periods. *Journal of Econometrics* 225:200–30. Themed Issue: Treatment Effect 1.

Casamatta, C. 2003. Financing and advising: Optimal financial contracts with venture capitalists 58:2059–85.

Casamatta, C., and C. Haritchabalet. 2007. Experience, screening and syndication in venture capital investments. *Journal of Financial Intermediation* 16:368–98.

Chemmanur, T. J., and Z. Chen. 2014. Venture capitalists versus angels: The dynamics of private firm financing contracts. *Review of Corporate Finance Studies* 3:39–86.

Eisfeldt, A. L., and D. Papanikolaou. 2013. Organization capital and the cross-section of expected returns. *The Journal of Finance* 68:1365–406.

Fershtman, C., and N. Gandal. 2011. Direct and indirect knowledge spillovers: the "social network" of open-source projects. *The RAND Journal of Economics* 42:70–91.

Gompers, P. A. 1995. Optimal investment, monitoring, and the staging of venture capital 50:1461–89.

Gompers, P. A., W. Gornall, S. N. Kaplan, and I. A. Strebulaev. 2020. How do venture capitalists make decisions? *Journal of Financial Economics* 135:169–90.

Gorman, M., and W. A. Sahlman. 1989. What do venture capitalists do? *Journal of Business Venturing* 4:231–48.

Gormley, T. A., and D. A. Matsa. 2011. Growing out of trouble? corporate responses to liability risk. *The Review of Financial Studies* 24:2781–821.

Hellmann, T., and M. Puri. 2002. Venture capital and the professionalization of start-up firms: Empirical evidence 57:169–97.

Hsu, D. H. 2004. What do entrepreneurs pay for venture capital affiliation? *The Journal of Finance* 59:1805–44.

Ibrahim, D. M. 2008. The (not so) puzzling behavior of angel investors. *Vanderbilt Law Review* 61:1405–52.

Kang, J.-K., Y. Li, and S. Oh. 2022. Venture capital coordination in syndicates, corporate monitoring, and firm performance. *Journal of Financial Intermediation* 50:100948–.

Kaplan, S. N., and P. Strömberg. 2003. Financial contracting theory meets the real world: An empirical analysis of venture capital contracts 70:281–315.

Krasner, H. 2022. Cost of poor software quality in the U.S. : A 2022 report. Working Paper, Consortium for Information & Software Quality (CISQ). Accessed: March 25, 2025.

Lerner, J. 1994. The syndication of venture capital investments. *Financial Management* 23:16–27.

———. 1995. Venture capitalists and the oversight of private firms. *Journal of Finance* 50:301–18.

Letouzey, J.-L. 2012. The SQALE method for evaluating technical debt. In *2012 Third International Workshop on Managing Technical Debt (MTD)*, 31–6. Zurich, Switzerland. doi:10.1109/MTD.2012.6225997.

McKinsey, and Co. 2020. Tech debt: Reclaiming tech equity. *McKinsey & Company* Accessed: March 25, 2025.

———. 2023. Breaking technical debt's vicious cycle to modernize your business. https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/breaking-technical-debts-vicious-cycle-to-modernize-your-business. Accessed: March 25, 2025.

Nagle, F. 2019. Open source software and firm productivity. *Management Science* 65:1191–215.

Peters, R. H., and L. A. Taylor. 2017. Intangible capital and the investment-q relation. *Journal of Financial Economics* 123:251–72.

Sculley, D., G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young. 2014. Machine learning: The high interest credit card of technical debt. In *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*.

Sculley, D., G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. 2015. Hidden technical debt in machine learning systems. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, 2503–2511. Cambridge, MA, USA: MIT Press.

Simonsohn, U., J. P. Simmons, and L. D. Nelson. 2020. Specification curve analysis. *Nature Human Behaviour* 4:1208–14.

Sun, L., and S. Abraham. 2021. Estimating dynamic treatment effects in event studies with heterogeneous treatment effects. *Journal of Econometrics* 225:175–99. Themed Issue: Treatment Effect 1.

U.S. Bureau of Labor Statistics. 2025. Software developers, quality assurance analysts, and testers. https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm. Median pay: $63.20 per hour (May 2024 data).

Appendix

A Variable Definitions

Table 16: Data and Variable Definitions

Variable	Description
A. Dimension of the data	
Firms	Open-core technology companies that commercialize open-source repositories and for which we observe longitudinal code histories, funding events, and employment data. Sample comprises 649 unique firms identified through OSSrank, a leading analytics platform for the open-source ecosystem.
Repositories	Core product repositories associated with each firm. We track the full commit history of 639 repositories hosted on GitHub or GitLab platforms during the period 2010–2025. A repository functions as the central storage location for the code underlying a firm's core product.
Month	Monthly observations covering the period 2010–2025.
Unit of observation	Firm–month panel combining repository-level code metrics with funding and employment information.
B. Dependent variables (Engineering quality metrics)	
Critical Violations	Number of SonarQube rule violations at the critical severity level per 1,000 lines of code in month t. Critical Violations correspond to the highest severity level, denoting issues with a significant probability of severe unintended consequences on the application, such as production crashes or major security flaws.
Violations (All)	Total number of SonarQube rule violations across all severity levels per 1,000 lines of code in month t. Violations denote the total number of deviations from engineering best practices (rules) in all states. Measures overall deviation from engineering best practices across reliability, security, and maintainability domains.

Continued on next page

Table 16 (continued)

Variable	Description
Maintainability Issues (All)	Total count of maintainability-related rule violations (all severities) per 1,000 lines of code. Maintainability Issues capture the total number of maintainability-related rule violations. Covers code patterns that make the codebase harder to modify or extend (e.g., code duplication, complex control flow).
Maintainability Issues (High)	Number of high-severity maintainability issues per 1,000 lines of code. High-Severity Maintainability Issues isolate only the most severe maintainability problems; those indicating substantial technical debt accumulation or elevated risk of code degradation. Captures major structural problems that materially hinder scalability and refactoring efforts.
Maintainability Issues (Low)	Number of low-severity maintainability issues per 1,000 lines of code. Low-Severity Maintainability Issues capture the more minor structural or readability deficiencies that still contribute to long-term maintainability costs rather than immediate functionality.
SonarQube Open Issues	Omitted from analysis. In retrospective scanning, every rule violation is simultaneously an open issue; the variable is therefore identical to Total Violations and is excluded to avoid double-counting.
Lines of code (LoC)	Total number of lines of source code in the firm's core repository at the end of month t.
SonarQube Security Hotspots	Number of security-sensitive code segments flagged by SonarQube as requiring manual audit, per 1,000 lines of code. Unlike rule violations which are confirmed deviations from best practices, Security Hotspots represent potential risk areas that require human audit to assess. Indicates density of potential (but not yet confirmed) security risks.
Bugs	Number of confirmed coding errors that inevitably lead to runtime failures or unexpected behavior, per 1,000 lines of code.

Continued on next page

Table 16 (continued)

Variable	Description
Reliability Issues (High)	Number of high-severity reliability issues per 1,000 lines of code. Critical defects that pose substantial risk to system stability and require rapid remediation, typically because they are likely to affect a large number of users or core functionality.
Reliability Issues (Low)	Number of low-severity reliability issues per 1,000 lines of code. Minor reliability risks or potential flaws in edge cases that have negligible immediate impact on the application's runtime performance but may degrade its robustness over time.
Technical Debt per Line of Code	Estimated dollar value of remediation effort per line of code. Let w denote the wage per hour ($55/hour) and let the SQALE Index be SonarQube's remediation effort for maintainability issues (in hours). Let ℓ denote the development time per line (in hours) and set $\ell = 0.5$ (30 minutes), SonarQube's default. We translate lines of code into a re-development effort proxy $\ell \times \text{LoC}$ hours and translate the raw SQALE Index into calibrated remediation effort using κ. We choose κ so that the median firm's calibrated technical debt equals a target share φ of median replacement-cost software assets (we set $\varphi = 0.20$): $$\kappa = \varphi \times \frac{\text{Median}(\ell \times \text{LoC})}{\text{Median}(\text{SQALE Index})}.$$ We then define $$\text{Technical Debt}_{i,t} = \kappa \times \text{SQALE Index}_{i,t} \times w,$$ and normalize by lines of code: $$\text{Technical Debt per LoC}_{i,t} = \frac{\text{Technical Debt}_{i,t}}{\text{LoC}_{i,t}}.$$

Continued on next page

Table 16 (continued)

Variable	Description
Technical Debt Ratio	Ratio of technical debt remediation effort to the estimated effort required to redevelop the codebase from scratch (Software Assets). Software analogue of a debt-to-assets ratio. This Sonar metric approximates the firm's software liabilities scaled by the "book value" of its software assets.
Technical Debt (level)	Total dollar value of the firm's technical debt in month t, defined as $\kappa \times \text{SQALE Index}_{i,t} \times w$ using the moment-matching calibration above. This yields a monthly, firm-level estimate of the replacement cost of the firm's software liabilities, denominated in USD.
C. Main explanatory variables	
$VC_{i,c,t}$ (Treated × Post)	Indicator equal to 1 if firm i in cohort c has obtained venture capital financing (first Seed or Series A round) at or before month t, and 0 otherwise. Main treatment variable in the difference-in-differences specification. Mathematically, this represents the interaction between the cohort-level treatment assignment and the post-event time indicators: $\text{Treated}_{i,c} \times \text{Post}_{i,c,t}$.
Treated	Dummy equal to 1 for firms that ever receive Seed or Series A venture funding during the sample period, and 0 for firms that remain untreated by the end of their respective event windows.
Post	Dummy equal to 1 for months at or after the first VC deal for treated firms (and the corresponding aligned event-time months for controls in each cohort), and 0 otherwise.
High Violations	Dummy equal to 1 if the firm's pre-treatment critical violation density (at $t = -1$) exceeds the treated-firm median, and 0 otherwise. Captures firms with worse baseline code quality—the "room for improvement" dimension.
Large Round	Dummy equal to 1 if the VC deal size (first Seed or Series A round amount, in USD) exceeds the treated-firm median, and 0 otherwise. Tests whether the magnitude of capital, rather than monitoring per se, drives quality improvements.

Continued on next page

Table 16 (continued)

Variable	Description
Technical Founder	Dummy equal to 1 if any member of the founding team (identified via LinkedIn job titles containing "founder" or "co-founder" at the time of first VC financing) simultaneously held a technical leadership role (CTO, VP of Engineering, Head of Engineering, or Director of Engineering) at the same firm. Captures whether the founding team includes individuals who combine entrepreneurial and engineering expertise.
Serial Entrepreneur	Dummy equal to 1 if any founder of the firm held a prior CEO, Chief, Founder, or President role at a different company before the first VC deal date (excluding assistant and vice president roles). Captures prior startup experience within the founding team.
Series B+ treatment (stage-specificity)	In the stage-specificity sample of mature firms, dummy equal to 1 in months at or after a Series B or later funding round (conditional on observing earlier Seed/Series A history), and 0 otherwise. Used to test whether capital injection alone drives quality improvements or if the effect is specific to early-stage VC involvement.
D. Other covariates and constructed variables	
Total firm headcount	Total number of employees associated with the firm based on LinkedIn data in month t.
Software engineer headcount	Number of employees with software-engineering roles (based on LinkedIn job titles) in month t.
New hires (24-month)	Total number of distinct new employees who started at the firm within 24 months of the first VC deal date and who were not active at the firm before funding. Identified from LinkedIn position records.
Senior engineering hires	Number of new hires (within 24 months post-deal) whose LinkedIn job titles indicate senior engineering roles: senior engineer/developer, staff engineer, principal engineer, architect, engineering director/VP, engineering manager, tech lead, or CTO.

Continued on next page

Table 16 (continued)

Variable	Description
Junior hires	Number of new hires (within 24 months post-deal) whose LinkedIn job titles indicate junior-level roles: junior, intern, entry-level, associate, graduate, or trainee positions.
Senior hire ratio	Ratio of senior engineering hires to total new hires in the 24-month post-deal window. Null for firms with zero new hires in the window.
Hired technical leader	Dummy equal to 1 if the firm hired a new CTO, VP of Engineering, or Head of Engineering within 24 months of the first VC deal.
New SWE hires	Number of new hires (within 24 months post-deal) with software engineering job titles: software engineer/developer, backend, frontend, full-stack, mobile developer, DevOps, SRE, platform engineer, data engineer, or ML engineer.
$X_{i,t}$	Vector of time-varying firm and project characteristics (e.g., pre-treatment code quality interactions, deal size interactions, measures of scale and growth) included as controls in augmented specifications.
Log LoC	Natural logarithm of lines of code in the firm's core repository.
Log total headcount	Natural logarithm of total firm headcount.
Log software engineer headcount	Natural logarithm of software engineer headcount.
Pre-treatment headcount growth	Difference in log average headcount between months $t \in [-4, -2]$ and $t \in [-12, -11]$ relative to the VC event month for firm i in cohort c. Used in propensity score matching and Mahalanobis distance matching to control for pre-treatment growth trajectories: $\text{Growth-pre}_{i,c} = \ln\left(\frac{1}{3}\sum_{t=-4}^{-2}\text{Headcount}_{i,c,t}\right) - \ln\left(\frac{1}{2}\sum_{t=-12}^{-11}\text{Headcount}_{i,c,t}\right)$

Continued on next page

Table 16 (continued)

Variable	Description
Pre-treatment SWE headcount growth	Analogous pre-treatment growth measure constructed using software engineer headcount instead of total headcount over the same pre-treatment windows.
Pre-treatment software growth	Analogous pre-treatment growth measure constructed using lines of code (LoC) over the same pre-treatment windows.
Software Assets	Estimated replacement cost of the codebase in month t. Software Assets represent a coarse approximation of the capital required to re-develop the codebase from scratch, defined using a replacement cost method: $\text{Software Assets}_{i,t} = \text{LoC}_{i,t} \times \text{Redevelopment Time per Line} \times \text{Hourly Wage}$ where Redevelopment Time represents the hours of labor required to produce a single line of production-ready code, calibrated using industry benchmarks on developer productivity.
Event-time indicators	Set of dummy variables for each quarter or month $t \in [-12, +12]$ relative to the VC event date. Used to estimate dynamic difference-in-differences (event-study) coefficients and test for parallel pre-trends.
Cohort × Firm FE ($\chi_{c,i}$)	Firm-by-cohort fixed effects that absorb time-invariant firm characteristics within each stacked cohort.
Month × Cohort FE ($\delta_{c,t}$)	Month-by-cohort fixed effects that control for common time-varying shocks affecting all firms within a cohort at month t.

Note: All engineering quality metrics are measured using SonarQube Community Edition static code analysis, which checks code against over 9,000 language-specific rules intended to operationalize engineering-quality heuristics and broadly aligned with software-quality frameworks (e.g., ISO/IEC 25010). Technical Debt calculations use the SQALE (Software Quality Assessment based on Lifecycle Expectations) methodology. The sample comprises 649 open-core technology firms with codebase histories spanning 2010–2025; complete histories are available for 639 of 734 identified repositories, yielding over 100,000 unique codebase snapshots. The stacked difference-in-differences design comprises 85 unique event cohorts with 225 treated firms and 330 controls, observing outcomes over a symmetric 24-month window ($t \in [-12, +12]$) around the first VC financing event.

B Language-Specific Analysis Configurations

Available in the Online Appendix.

C Stacked difference-in-differences: estimand and identification

Design. We observe firms $i = 1, \ldots, N$ over calendar months $t = 1, \ldots, T$. Let E_i denote the month in which firm i first receives VC funding (Seed, or Series A in the absence of Seed), with $E_i = \infty$ for never-funded firms. The objective is to estimate the causal effect of the first VC financing event in settings with staggered treatment timing. Motivated by the limitations of standard two-way fixed effects estimators under heterogeneous dynamic treatment effects, we use a stacked difference-in-differences design with cohort-specific clean controls, following the cohort-stacking approach in Gormley and Matsa (2011).

Setup: cohorts, controls, and event time. A *cohort* is the set of firms that first receive VC funding in the same calendar month. Let C collect all months c in which at least one firm is first funded:

$$C = \{c \in \{1, \ldots, T\} : E_i = c \text{ for some firm } i\}.$$

For each cohort month c, write $\mathcal{T}_c = \{i : E_i = c\}$ for the set of treated firms.

We study outcomes in an event-time window of κ months before and after the treatment event (we use $\kappa = 12$). For a firm treated at month c, event time $e = t - c$ measures months relative to treatment, so $e \in [-\kappa, \kappa]$, where $e < 0$ denotes pre-treatment months and $e \geq 0$ denotes treatment and post-treatment months.

For each cohort c, the control group consists of *clean controls* that remain untreated throughout cohort c's entire event window. Formally, cohort c's control set is

$$\mathcal{J}_c = \{j : E_j > c + \kappa\},$$

ensuring that control firms j do not receive funding until after we stop observing cohort c. This set includes both never-funded firms and firms that are funded after the event window; under Assumption NA below, both are untreated throughout $t \in [c - \kappa,\ c + \kappa]$.

We restrict attention to cohorts that admit a full event-time window and have nonempty clean-control sets. We retain only *admissible cohorts*:

$$C_\kappa = \left\{ c \in C : \begin{array}{l} \text{(i) full window observable: } T_{\min} + \kappa \leq c \leq T_{\max} - \kappa, \\ \text{(ii) clean controls exist: } |\mathcal{J}_c| \geq 1 \end{array} \right\}.$$

For each admissible cohort $c \in C_\kappa$, we form the cohort-specific sample $\mathcal{S}_c = \mathcal{T}_c \cup \mathcal{J}_c$ observed over months $t \in [c - \kappa,\ c + \kappa]$. We then *stack* these cohort-specific samples, indexing observations by (i, c, e) where $e = t - c$. A firm can appear in multiple stacked cohorts (as a control for several

earlier-treated cohorts), which is why observation counts differ between summary statistics and stacked regressions.

Causal framework: potential outcomes and identification. Let Y_{it}^{g} denote the potential outcome for firm i in month t if the firm's first VC funding occurs in month g (with $g = \infty$ for never-funded). The observed outcome is $Y_{it} = Y_{it}^{E_i}$. Under Assumption NA below, if $g > t$ then $Y_{it}^{g} = Y_{it}^{\infty}$, so potential outcomes coincide with the never-funded potential outcome prior to the funding month.

The treatment effect for firm i in cohort c at event time e is

$$\delta_{i,e}^{c} = Y_{i,c+e}^{c} - Y_{i,c+e}^{\infty}.$$

We target the average treatment effect on the treated at each cohort-event pair:

$$\theta_{c,e} = \mathbb{E}[\delta_{i,e}^{c} \mid E_i = c].$$

Identification rests on two assumptions applied within each cohort c:

Assumption PT (Parallel Trends). *Absent treatment, treated and control firms in cohort c would have evolved along parallel paths.* Formally, for all event times $e, e' \in [-\kappa, \kappa]$,

$$\mathbb{E}[Y_{i,c+e}^{\infty} - Y_{i,c+e'}^{\infty} \mid E_i = c] = \mathbb{E}[Y_{i,c+e}^{\infty} - Y_{i,c+e'}^{\infty} \mid i \in \mathcal{J}_c].$$

This states that the counterfactual evolution of outcomes (in the absence of treatment) is the same for treated and control firms.

Assumption NA (No Anticipation). *Receiving VC funding has no effect on outcomes before the funding month.* Formally, for all g and all $t < g$,

$$Y_{it}^{g} = Y_{it}^{\infty}.$$

Equivalently, for all $e < 0$, $\theta_{c,e} = 0$.

Under these assumptions, the usual difference-in-differences estimator identifies $\theta_{c,e}$. To see this, let $\bar{e} < 0$ denote a pre-treatment baseline period. The cohort-c DID estimand at event time $e \geq 0$ is:

$$\widehat{\theta}_{c,e}^{\mathrm{DID}} = \underbrace{\mathbb{E}[Y_{i,c+e} - Y_{i,c+\bar{e}} \mid E_i = c]}_{\text{treated change}} - \underbrace{\mathbb{E}[Y_{i,c+e} - Y_{i,c+\bar{e}} \mid i \in \mathcal{J}_c]}_{\text{control change}}.$$

Expanding the treated change using potential outcomes and assumption NA (which implies $Y_{i,c+\bar{e}}^{c} =$

$Y^{\infty}_{i,c+\bar{e}}$ for $\bar{e} < 0$):

$$\begin{aligned}\mathbb{E}[Y_{i,c+e} - Y_{i,c+\bar{e}} \mid E_i = c] &= \mathbb{E}[Y^{c}_{i,c+e} - Y^{\infty}_{i,c+\bar{e}} \mid E_i = c] \\ &= \mathbb{E}[Y^{\infty}_{i,c+e} + \delta^{c}_{i,e} - Y^{\infty}_{i,c+\bar{e}} \mid E_i = c] \\ &= \theta_{c,e} + \mathbb{E}[Y^{\infty}_{i,c+e} - Y^{\infty}_{i,c+\bar{e}} \mid E_i = c].\end{aligned}$$

The control change directly estimates $\mathbb{E}[Y^{\infty}_{i,c+e} - Y^{\infty}_{i,c+\bar{e}} \mid i \in \mathcal{J}_c]$. Under PT, these two counterfactual trends coincide, so

$$\widehat{\theta}^{\text{DID}}_{c,e} = \theta_{c,e}.$$

Aggregation: from cohort-event effects to summary parameters. Each cohort-event pair (c, e) yields its own treatment effect $\theta_{c,e}$. To summarize these into a single time-path, we average across cohorts at each event time, weighting by cohort size:

$$\tau_e = \sum_{c \in \mathcal{C}_\kappa} \omega_c \, \theta_{c,e}, \qquad \omega_c = \frac{|\mathcal{T}_c|}{\sum_{c' \in \mathcal{C}_\kappa} |\mathcal{T}_{c'}|}.$$

This gives an event-time profile $\{\tau_e : e \in [-\kappa, \kappa]\}$ representing the average effect across all cohorts at each lag/lead.

For a single scalar summary of the post-treatment effect, we average over post-treatment event times:

$$\tau_{\text{post}} = \frac{1}{\kappa + 1} \sum_{e=0}^{\kappa} \tau_e.$$

Regression implementation. In the stacked dataset indexed by (i, c, e), define

$$\text{Treat}_{ic} = \mathbf{1}\{E_i = c\}, \qquad \text{Post}_{ce} = \mathbf{1}\{e \geq 0\}.$$

The stacked DID regression is

$$Y_{ice} = \beta \cdot (\text{Treat}_{ic} \times \text{Post}_{ce}) + \alpha_{ic} + \lambda_{ce} + \Gamma' X_{it} + \epsilon_{ice},$$

where α_{ic} are cohort-by-firm fixed effects, λ_{ce} are cohort-by-event-time fixed effects, and X_{it} are time-varying firm controls. The coefficient β estimates τ_{post} under assumptions PT and NA. Standard errors are clustered by cohort and firm to account for within-firm serial correlation and within-cohort dependence in the stacked sample.

To recover dynamic treatment effects, we estimate an event-study specification that replaces the

single post indicator with event-time indicators and omits the event-time zero category as the reference period. In the empirical implementation and figures, event time is discretized into quarters $q(e)$. The quarter mapping groups months $\{-4, -3, -2\}$ into $q = 0$, months $\{-1, 0, 1\}$ into $q = 1$, and so on. After a re-indexing shift, $q = 0$ serves as the omitted reference category. Because $q = 0$ contains only pre-treatment months, the reference level is cleanly pre-treatment. The first post-reference quarter $q = 1$ pools one pre-treatment month ($e = -1$) with two post-treatment months ($e = 0, 1$), which attenuates the estimated treatment onset; all subsequent quarters are fully post-treatment.

$$Y_{ice} = \sum_{\substack{e \in [-\kappa, \kappa] \\ e \neq 0}} \beta_e \cdot (\text{Treat}_{ic} \times \mathbf{1}\{\text{event time} = e\}) + \alpha_{ic} + \lambda_{ce} + \Gamma' X_{it} + \epsilon_{ice}.$$

Each β_e estimates τ_e (with $\tau_0 = 0$ by construction as the reference point). Pre-treatment coefficients $\{\beta_e : e < 0\}$ provide a visual test of the no-anticipation and parallel-trends assumptions: if these assumptions hold, pre-treatment coefficients should be statistically indistinguishable from zero.

D Matching on Pre-Treatment Headcount Growth

Let firms be indexed by $i = 1, \ldots, N$ and time by $t = 1, \ldots, T$. Let M_{it} be the metric of interest.

Define each firm's cohort (the month in which it received its first Seed or Series A round) as

$$E_i \in \{1, \ldots, T\} \cup \{\infty\},$$

where $E_i = c < \infty$ means firm i is first treated in period c, and $E_i = \infty$ means it is never treated.

Let

$$C = \left\{c : \exists i \text{ with } E_i = c\right\}$$

denote the set of treatment cohorts, and for each cohort $c \in C$ let

$$\mathcal{T}_c = \{i : E_i = c\}$$

be the set of firms treated in cohort c.

For each cohort c, define the set of pre-treatment periods used to construct growth histories. We use two windows: an early window $\{-12, -11\}$ and a late window $\{-4, -3, -2\}$, corresponding to the intervals $t \in [-12, -11]$ and $t \in [-4, -2]$ described in the main text.

Let g_{it} denote firm i's growth measure in period t (growth of total headcount, software developer headcount or software). For each firm i and cohort c, aggregate and stack these pre-treatment metrics

into a vector

$$G_i^c = \left(g_{it}\right)_{t \in \mathcal{T}^{\text{pre}}(c)} \in \mathbb{R}^p,$$

where p is the number of growth metrics we are using for the specification.

For each cohort c, define the set of eligible control firms $\mathcal{J}_c$; these are firms that are not yet treated by the end of the event window for this cohort $c + 12$ (or possibly never treated).

$$\mathcal{J}_c \subseteq \{j : E_j \geq c\}.$$

Estimate the covariance matrix of pre-treatment growth histories among the candidate controls:

$$\widehat{\Sigma}_c = \text{Var}\big(G_j^c : j \in \mathcal{J}_c\big).$$

The Mahalanobis distance in the space of pre-treatment growth between any treated firm $i \in \mathcal{T}_c$ and control $j \in \mathcal{J}_c$ is

$$d_{M,c}(i, j) = \sqrt{\big(G_i^c - G_j^c\big)' \widehat{\Sigma}_c^{-1} \big(G_i^c - G_j^c\big)}.$$

Since for this specification we match on the pre-treatment growth of headcount only, $\widehat{\Sigma}_c^{-1}$ reduces to $1/\sigma_c^2$.

For each treated firm $i \in \mathcal{T}_c$, define its set of matched controls as its K nearest neighbors in Mahalanobis distance among eligible controls:

$$\mathcal{M}_c(i) = \arg\min_{\substack{S \subseteq \mathcal{J}_c \\ |S| = K}} \sum_{j \in S} d_{M,c}(i, j).$$

For one-to-one matching, set $K = 1$ and $\mathcal{M}_c(i) = \{j^*(i)\}$ with

$$j^*(i) = \arg\min_{j \in \mathcal{J}_c} d_{M,c}(i, j).$$

Imposing a caliper adds the restriction $d_{M,c}(i, j) \leq \delta$ for some $\delta > 0$.

The set of matched controls for cohort c is then

$$C_c^{\text{match}} = \bigcup_{i \in \mathcal{T}_c} \mathcal{M}_c(i).$$

The estimation sample for cohort c ("cohort-c stack") is

$$\mathcal{S}_c = \mathcal{T}_c \cup C_c^{\text{match}}.$$

In our primary specification, we employ centroid matching rather than individual nearest-neighbor matching. Specifically, for each cohort c we compute the treated-group centroid $\bar{G}_c^{\mathcal{T}} = |\mathcal{T}_c|^{-1} \sum_{i \in \mathcal{T}_c} G_i^c$ and select as matched controls the K eligible firms closest to this centroid in Mahalanobis distance, subject to caliper δ:

$$C_c^{\text{match}} = \arg \min_{\substack{S \subseteq \mathcal{J}_c,\, |S|=K \\ d_{M,c}(\bar{G}_c^{\mathcal{T}}, j) \leq \delta \;\forall\, j \in S}} \sum_{j \in S} d_{M,c}\big(\bar{G}_c^{\mathcal{T}},\, j\big).$$

This yields a comparison group whose pre-treatment growth trajectories resemble those of the typical VC-backed startup in the cohort, rather than matching each treated firm individually. We set $\delta = 1$ standard deviation of the within-cohort Mahalanobis distance distribution. Appendix E demonstrates that our results are robust to alternative matching configurations, including individual nearest-neighbor matching with $K \in \{1, 3\}$, additional matching features (software engineering headcount, lines of code), and calipers ranging from 0.5 to 2.0 standard deviations.

E Matching Sensitivity Analysis

This appendix examines the sensitivity of the matched difference-in-differences estimates to the choice of matching configuration. We construct a specification curve (Simonsohn, Simmons, and Nelson, 2020) over 72 configurations that vary along three dimensions: **(i)** the feature set used to compute Mahalanobis distances (headcount growth only; headcount plus lines of code; headcount plus software engineering headcount; all three); **(ii)** the matching mode (centroid matching, individual nearest-neighbor with $K = 1$, and individual nearest-neighbor with $K = 3$); and **(iii)** the caliper, defined as a multiple of the within-cohort standard deviation of Mahalanobis distances ($\delta \in \{0.5, 0.75, 1.0, 1.25, 1.5, 2.0\}$). Matching is performed without replacement within each cohort; controls may appear across different cohorts but are used at most once per cohort. For each configuration, we re-estimate the stacked difference-in-differences specification from Section C on the resulting matched sample.

Figures 12–15 present the specification curves for our four core engineering-quality outcomes. In each figure, the top panel plots the point estimate and 95% confidence interval for every specification, sorted by coefficient magnitude. The solid black line marks the baseline (unmatched) estimate and its 95% confidence band (gray shading); the dashed red line marks zero. The red diamond identifies our primary matched specification (centroid matching on headcount growth, caliper $\delta = 1$). The bottom panel displays an indicator matrix identifying the feature set, matching mode, and caliper active in each specification.

Sign consistency is strong: 100% of critical-violation coefficients, 97% of total-violation coefficients, 99% of maintainability-issue coefficients, and 93% of technical-debt coefficients are negative. At conventional significance levels, 25% of critical-violation specifications, 53% of total-violation

specifications, 88% of maintainability-issue specifications, and 14% of technical-debt specifications reject the null at the 5% level. The drop in statistical significance relative to the baseline is largely a consequence of the substantial reduction in sample size induced by matching: the primary matched sample retains approximately 46,000 of 325,000 baseline observations (an 86% reduction), and individual nearest-neighbor configurations ($K = 1$, $K = 3$) can reduce the sample to fewer than 5,000 observations. Despite this loss of precision, coefficient magnitudes remain stable—the median matched-to-baseline coefficient ratio ranges from 0.93 to 1.20 across outcomes—indicating that the attenuation in significance reflects reduced statistical power rather than a weakening of the underlying treatment effect. The primary specification falls near the median of the coefficient distribution across all four outcomes, confirming that our baseline matched estimates are not driven by a particular choice of matching parameters.

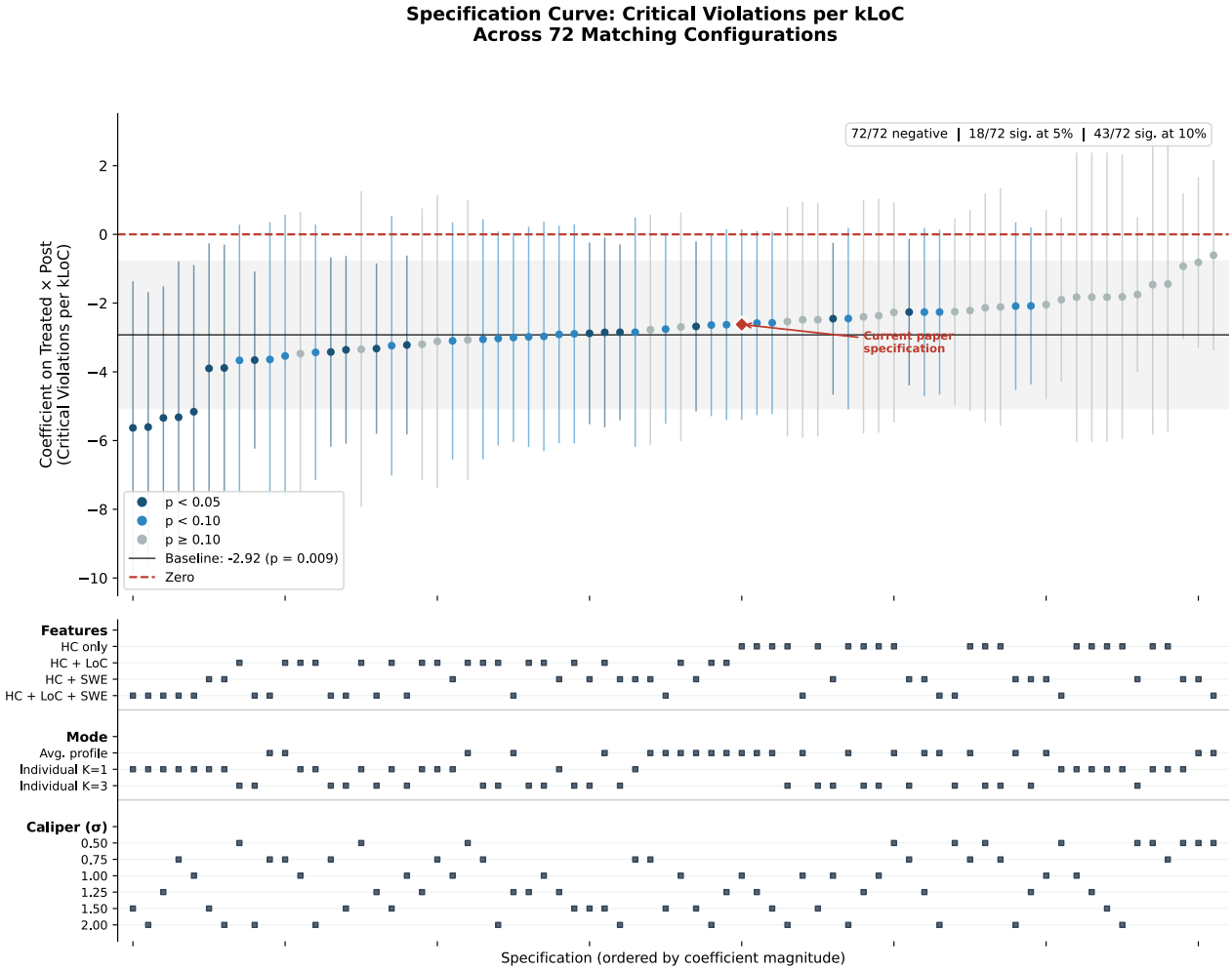


Figure 12: Specification Curve: Critical Violations per 1,000 LoC. The top panel plots the estimated treatment effect and 95% confidence interval across 72 matching configurations, sorted by coefficient magnitude. The solid black line marks the baseline (unmatched) estimate; the dashed red line marks zero. The bottom panel indicates the active feature set, matching mode, and caliper for each specification.

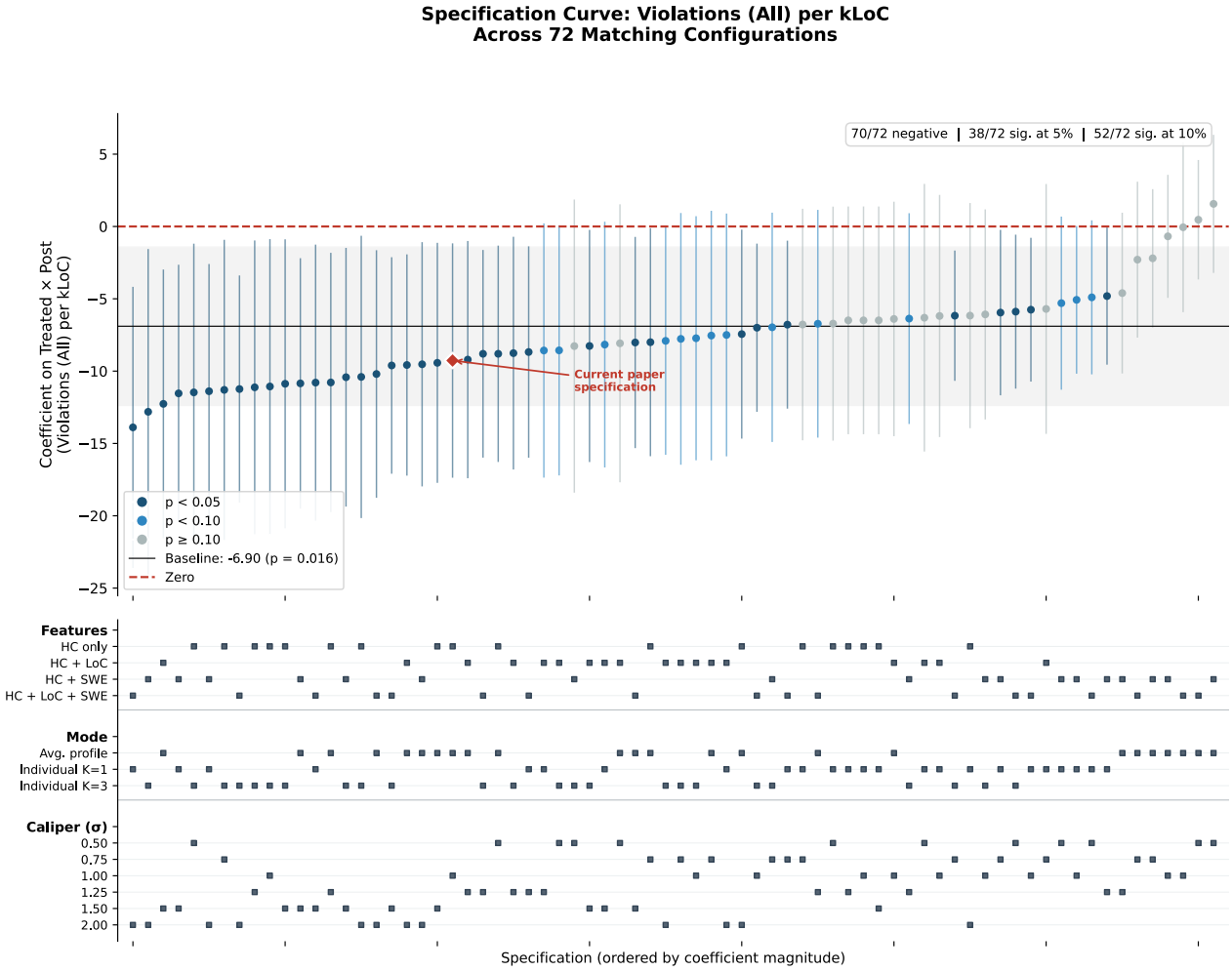


Figure 13: Specification Curve: Violations (All) per 1,000 LoC. See Figure 12 for description of the layout.

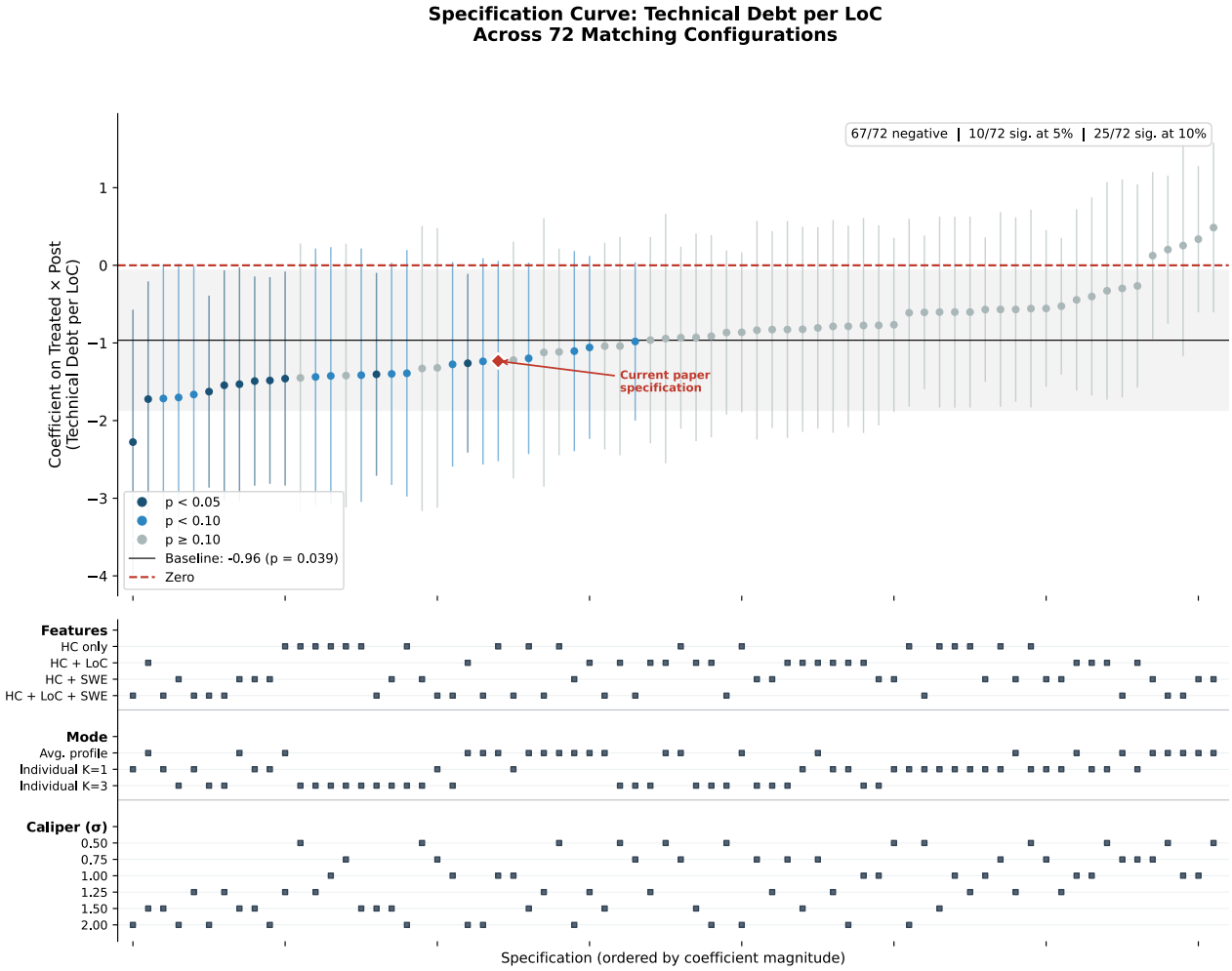


Figure 14: Specification Curve: Technical Debt per Line of Code. See Figure 12 for description of the layout.

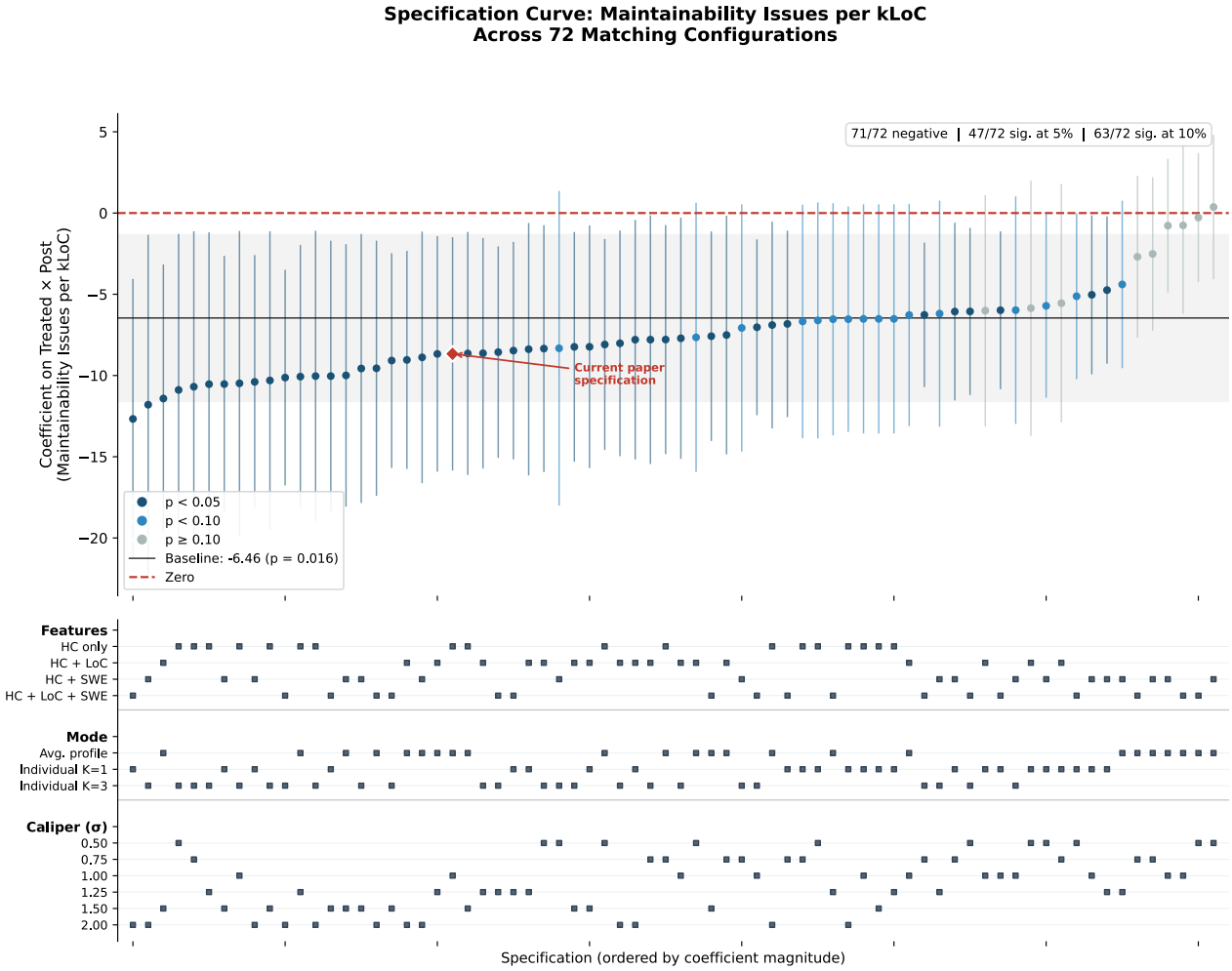


Figure 15: Specification Curve: Maintainability Issues per 1,000 LoC. See Figure 12 for description of the layout.

F Founding-Team Heterogeneity

We supplement our main heterogeneity analysis with two additional dimensions based on founding-team characteristics constructed from LinkedIn career history data. For each treated firm, we match the firm to its LinkedIn company profile and identify individuals whose job titles contain "founder" or "co-founder" and who were active at the firm within six months of the first VC deal date. We then retrieve the full career history of each identified founder to construct two binary indicators.

Technical Founder equals one if any member of the founding team simultaneously held a technical leadership role—CTO, VP of Engineering, Head of Engineering, or Director of Engineering—at the same firm around the deal date. This captures whether the founding team includes individuals who combine entrepreneurial and engineering expertise, which may moderate the effect of VC-induced professionalization if such founders have already established engineering discipline prior to funding.

Serial Entrepreneur equals one if any founder previously held a CEO, Chief Officer, Founder, or President role at a different company before the first VC deal date. Serial entrepreneurs may bring established engineering practices and organizational templates from prior ventures, potentially attenuating the marginal impact of VC oversight.

We interact each indicator with the Treated $\times$ Post term, following the same triple-interaction specification used for the pre-treatment quality and deal size splits in the main analysis. Tables 17–19 report the full regression output. Each table presents eight columns: columns (1)–(4) correspond to one outcome, and columns (5)–(8) to a second related outcome, with each set of four columns cycling through the Large Codebase, Static Typed Language, Technical Founder, and Serial Entrepreneur interaction specifications in that order.

Table 17: Founder Heterogeneity: Technical Debt per LoC and Technical Debt Ratio

	Technical Debt per LoC				**Technical Debt Ratio**			
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Treated × Post	−2.483**	−1.250*	−0.966*	−1.080	−0.252**	−0.130*	−0.100*	−0.112
	(1.144)	(0.731)	(0.568)	(0.745)	(0.114)	(0.073)	(0.056)	(0.074)
Post × Large Codebase (=1)	−0.446				−0.046			
	(0.422)				(0.042)			
Treated × Post × Large Codebase (=1)	2.384**				0.239**			
	(1.175)				(0.117)			
Post × Static Typed (=1)		−0.189				−0.019		
		(0.256)				(0.026)		
Treated × Post × Static Typed (=1)		0.747				0.079		
		(0.863)				(0.086)		
Post × Technical Founder (=1)			0.365				0.035	
			(0.426)				(0.043)	
Treated × Post × Technical Founder (=1)			−0.277				−0.026	
			(0.997)				(0.100)	
Post × Serial Entrepreneur (=1)				0.093				0.009
				(0.331)				(0.033)
Treated × Post × Serial Entrepreneur (=1)				0.177				0.021
				(0.973)				(0.097)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	321,291	325,029	325,029	325,029	321,291	325,029	325,029	325,029
R^2	0.776	0.777	0.777	0.777	0.776	0.776	0.776	0.776

This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$. Columns (1) and (5) interact with a Large Codebase indicator (above-median lines of code). Columns (2) and (6) interact with a Static Typed Language indicator. Columns (3) and (7) interact with a Technical Founder indicator (founding team includes a simultaneous CTO/VP Eng/Director Eng role). Columns (4) and (8) interact with a Serial Entrepreneur indicator (any founder previously held a C-suite or founder role at a prior firm). See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses, clustered by cohort and firm. * p<0.1, ** p<0.05, *** p<0.01.

Table 18: Founder Heterogeneity: Violations per LoC and Critical Violations per LoC

	Violations per LoC				Critical Violations per LoC			
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Treated × Post	−15.499**	−8.295*	−5.960*	−7.602*	−5.653**	−3.129**	−2.910**	−3.270*
	(7.086)	(4.363)	(3.185)	(4.123)	(2.674)	(1.346)	(1.282)	(1.647)
Post × Large Codebase (=1)	−0.325				−1.115			
	(1.782)				(0.690)			
Treated × Post × Large Codebase (=1)	13.456*				4.306			
	(7.164)				(2.820)			
Post × Static Typed (=1)		−0.867				−0.131		
		(1.337)				(0.483)		
Treated × Post × Static Typed (=1)		3.649				0.538		
		(5.186)				(2.738)		
Post × Technical Founder (=1)			1.508				0.643	
			(2.043)				(0.895)	
Treated × Post × Technical Founder (=1)			−4.416				−0.538	
			(6.463)				(2.241)	
Post × Serial Entrepreneur (=1)				0.527				−0.391
				(1.543)				(0.704)
Treated × Post × Serial Entrepreneur (=1)				1.122				1.055
				(5.617)				(2.281)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	321,291	325,029	325,029	325,029	321,291	325,029	325,029	325,029
R^2	0.816	0.817	0.817	0.817	0.823	0.820	0.820	0.820

This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$. Columns (1) and (5) interact with a Large Codebase indicator. Columns (2) and (6) interact with a Static Typed Language indicator. Columns (3) and (7) interact with a Technical Founder indicator. Columns (4) and (8) interact with a Serial Entrepreneur indicator. Violations per LoC counts total rule violations across all severity levels per 1,000 lines of code. Critical Violations per LoC counts only the highest-severity violations per 1,000 lines of code. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses, clustered by cohort and firm. * p<0.1, ** p<0.05, *** p<0.01.

Table 19: Founder Heterogeneity: Bugs per LoC and Reliability Issues (High) per LoC

	Bugs per LoC				**Reliability Issues (High) per LoC**			
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Treated × Post	−1.061	−0.341	−0.182	−0.411	−0.577	−0.101	−0.267	−0.355
	(0.971)	(0.531)	(0.334)	(0.419)	(0.411)	(0.124)	(0.190)	(0.246)
Post × Large Codebase (=1)	−0.215				−0.242			
	(0.529)				(0.227)			
Treated × Post × Large Codebase (=1)	0.932				0.545			
	(0.961)				(0.404)			
Post × Static Typed (=1)		0.071				−0.103		
		(0.226)				(0.095)		
Treated × Post × Static Typed (=1)		−0.307				−0.262		
		(0.713)				(0.356)		
Post × Technical Founder (=1)			0.061				−0.003	
			(0.270)				(0.039)	
Treated × Post × Technical Founder (=1)			−1.012				0.146	
			(0.953)				(0.166)	
Post × Serial Entrepreneur (=1)				−0.020				−0.032
				(0.204)				(0.046)
Treated × Post × Serial Entrepreneur (=1)				−0.089				0.306
				(0.725)				(0.235)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	321,291	325,029	325,029	325,029	321,291	325,029	325,029	325,029
R^2	0.833	0.833	0.833	0.833	0.740	0.740	0.740	0.740

This table reports the coefficient β from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$. Columns (1) and (5) interact with a Large Codebase indicator. Columns (2) and (6) interact with a Static Typed Language indicator. Columns (3) and (7) interact with a Technical Founder indicator. Columns (4) and (8) interact with a Serial Entrepreneur indicator. Bugs per LoC counts coding errors that inevitably lead to runtime failures per 1,000 lines of code. Reliability Issues (High) per LoC counts high-severity reliability defects posing significant risk to system stability per 1,000 lines of code. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses, clustered by cohort and firm. * p<0.1, ** p<0.05, *** p<0.01.

Interpretation. Across all three outcome groupings, the triple-interaction terms for Technical Founder and Serial Entrepreneur are uniformly statistically insignificant, and the point estimates are small relative to the baseline Treated × Post coefficients. For technical debt, the interaction term Treated × Post × Technical Founder (=1) is −0.277 (s.e. 0.997) and the corresponding Serial En-

trepreneur interaction is 0.177 (s.e. 0.973)—both close to zero and far from conventional significance thresholds. The pattern is equally flat for violations, critical violations, bugs, and reliability issues.

This constitutes an informative null result. The professionalization effect of VC investment—the reduction in technical debt, rule violations, and reliability defects documented in the main analysis—is not concentrated among firms whose founders lack technical or entrepreneurial experience. Both technically sophisticated founding teams and first-time founders exhibit similar post-investment improvements in engineering quality. This suggests that the VC effect operates through channels beyond the direct augmentation of founder human capital, such as the introduction of formal engineering governance or changes in organizational incentives that apply regardless of the founding team's prior experience.

Post-Funding Hiring Composition

We extend the heterogeneity analysis to examine whether post-funding changes in team composition moderate the treatment effect. Using LinkedIn employment data, we construct two hiring-composition indicators for each treated firm. *Hired Technical Leader* equals one if the firm hired a CTO, VP of Engineering, Head of Engineering, or Director of Engineering within six months of the first VC deal date. *High Senior Ratio* equals one if the ratio of senior-level engineering hires among all new engineering employees exceeds the treated-firm median. We interact each indicator with the Treated $\times$ Post term in a triple-interaction specification.

Tables 20–21 report the full regression output. Each table presents three columns per outcome: Column (1) reproduces the baseline, Column (2) introduces the hired-technical-leader interaction, and Column (3) introduces the high-senior-ratio interaction.

Table 20: Hiring-Composition Heterogeneity: Critical Violations and Total Violations per kLoC

	Critical Violations / kLoC			**Violations / kLoC**		
	(1)	(2)	(3)	(4)	(5)	(6)
Treated × Post	−2.923***	−3.630**	−2.314**	−6.897**	−8.959**	−6.579*
	(1.094)	(1.801)	(1.091)	(2.795)	(4.088)	(3.692)
Post × Hired Technical Leader		0.378			−0.027	
		(0.941)			(1.709)	
Treated × Post × Hired Technical Leader		1.261			4.626	
		(2.376)			(5.419)	
Post × High Senior Ratio			0.947			1.922
			(0.737)			(1.486)
Treated × Post × High Senior Ratio			−2.256			−2.376
			(2.850)			(5.710)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	325,029	325,029	325,029	325,029	325,029	325,029
R^2	0.820	0.820	0.820	0.817	0.817	0.817

This table reports triple-interaction specifications examining whether post-funding hiring composition moderates the treatment effect. Hired Technical Leader is an indicator equal to one if the firm hired a CTO, VP of Engineering, or equivalent within six months of funding. High Senior Ratio is an indicator equal to one if the ratio of senior engineering hires exceeds the treated-firm median. Columns (1) and (4) reproduce the baseline. Critical Violations and Violations are measured per 1,000 lines of code. *Note:* Standard errors in parentheses, clustered by cohort and firm. * p<0.1, ** p<0.05, *** p<0.01.

Table 21: Hiring-Composition Heterogeneity: Technical Debt per LoC and Maintainability Issues per kLoC

	TD per LoC ($)			Maint Issues / kLoC		
	(1)	(2)	(3)	(4)	(5)	(6)
Treated × Post	−0.965**	−1.406*	−1.086	−6.461**	−8.381**	−6.554*
	(0.460)	(0.733)	(0.684)	(2.621)	(3.979)	(3.652)
Post × Hired Technical Leader		0.086			0.525	
		(0.323)			(1.659)	
Treated × Post × Hired Technical Leader		0.911			3.845	
		(0.879)			(4.980)	
Post × High Senior Ratio			0.229			2.305
			(0.296)			(1.407)
Treated × Post × High Senior Ratio			0.096			−1.716
			(0.903)			(5.122)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	325,029	325,029	325,029	325,029	325,029	325,029
R^2	0.777	0.777	0.777	0.790	0.790	0.790

This table reports triple-interaction specifications examining whether post-funding hiring composition moderates the treatment effect. Hired Technical Leader is an indicator equal to one if the firm hired a CTO, VP of Engineering, or equivalent within six months of funding. High Senior Ratio is an indicator equal to one if the ratio of senior engineering hires exceeds the treated-firm median. Columns (1) and (4) reproduce the baseline. Technical Debt per LoC is measured in dollars. Maintainability Issues are measured per 1,000 lines of code. *Note:* Standard errors in parentheses, clustered by cohort and firm. * p<0.1, ** p<0.05, *** p<0.01.

Across all four outcomes and both hiring-composition dimensions, the triple-interaction terms are uniformly insignificant, and the base treatment effect retains significance in every specification. For critical violations, the hired-technical-leader interaction is +1.261 (SE = 2.376) and the high-senior-ratio interaction is −2.256 (SE = 2.850). Neither approaches conventional significance. The pattern is consistent across total violations, technical debt, and maintainability issues. This hiring-composition null complements the founding-team null reported above: the VC professionalization effect is not moderated by observable human-capital characteristics of either the founding team or the post-funding hires.

G Level Outcomes: Event-Study Figures

This appendix presents the dynamic difference-in-differences estimates for level (log count) outcomes. Unlike the per-LoC density metrics reported in the main text, these specifications use the natural logarithm of the absolute count of violations, critical violations, maintainability issues, and technical debt (in dollars) as dependent variables.

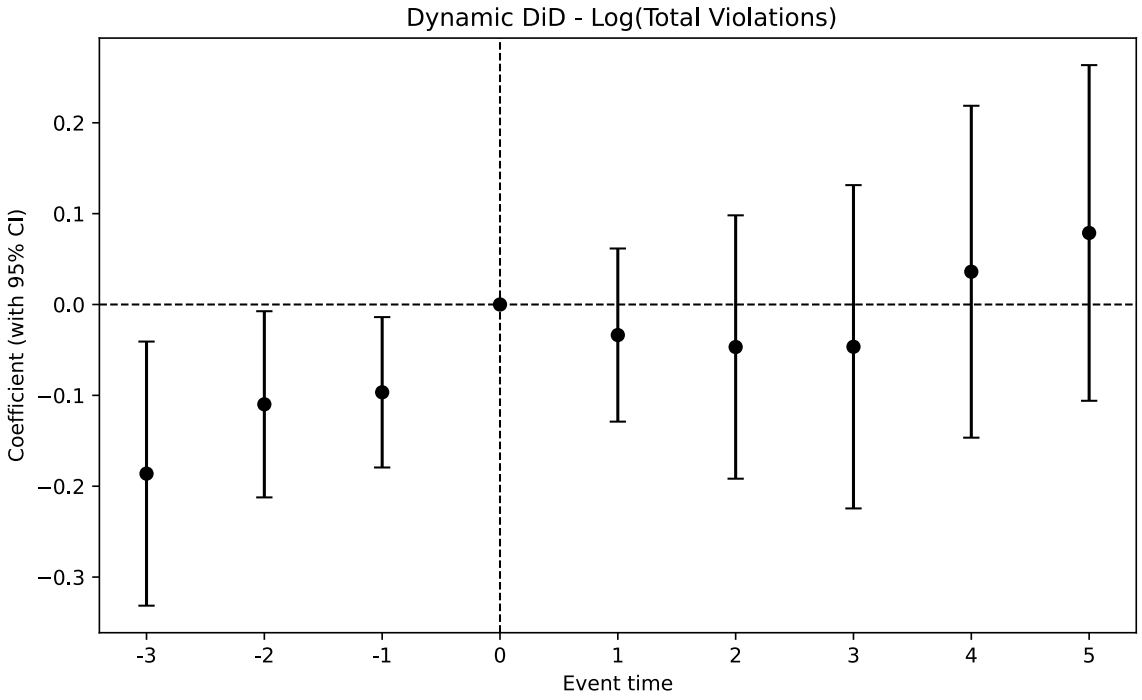


Figure 16: Dynamic DiD Estimate — Ln(Total Violations)

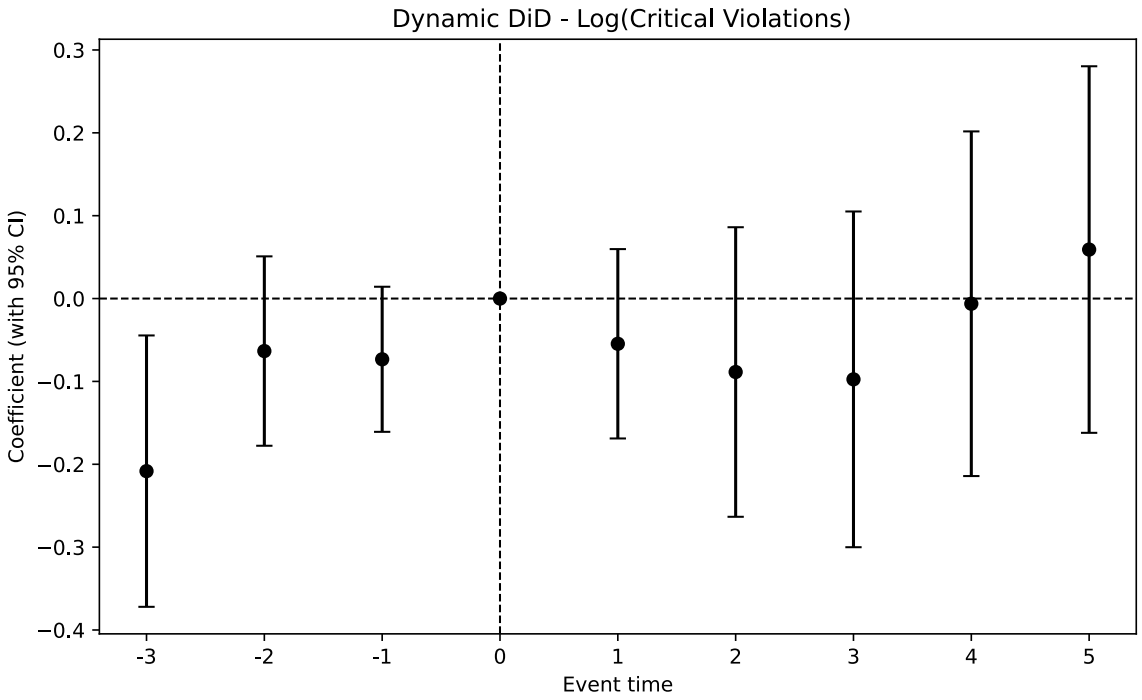


Figure 17: Dynamic DiD Estimate — Ln(Critical Violations)

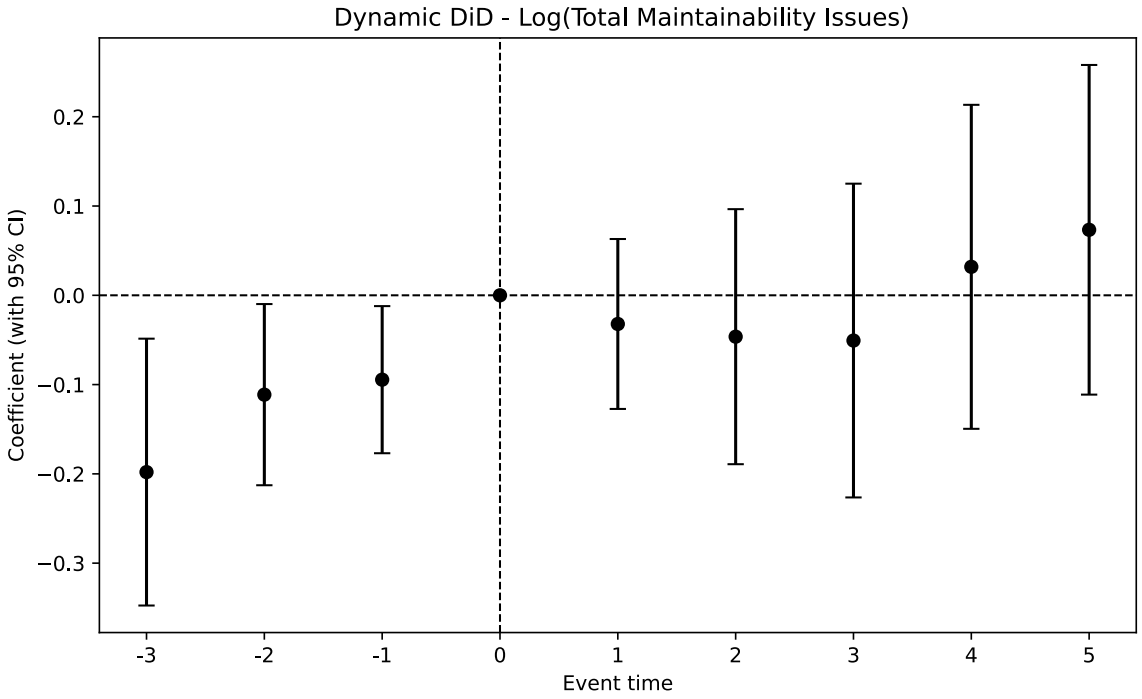


Figure 18: Dynamic DiD Estimate — Ln(Total Maintainability Issues)

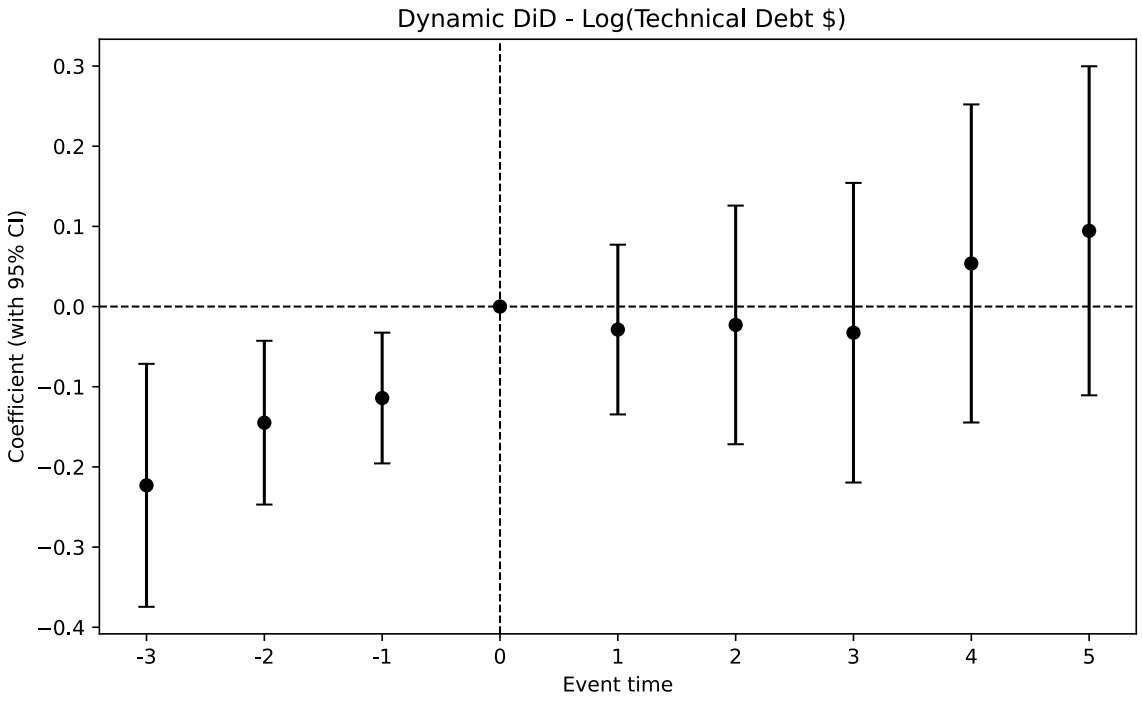


Figure 19: Dynamic DiD Estimate — Ln(Technical Debt $)

H Stage-Specificity Estimates: Later Financing Rounds

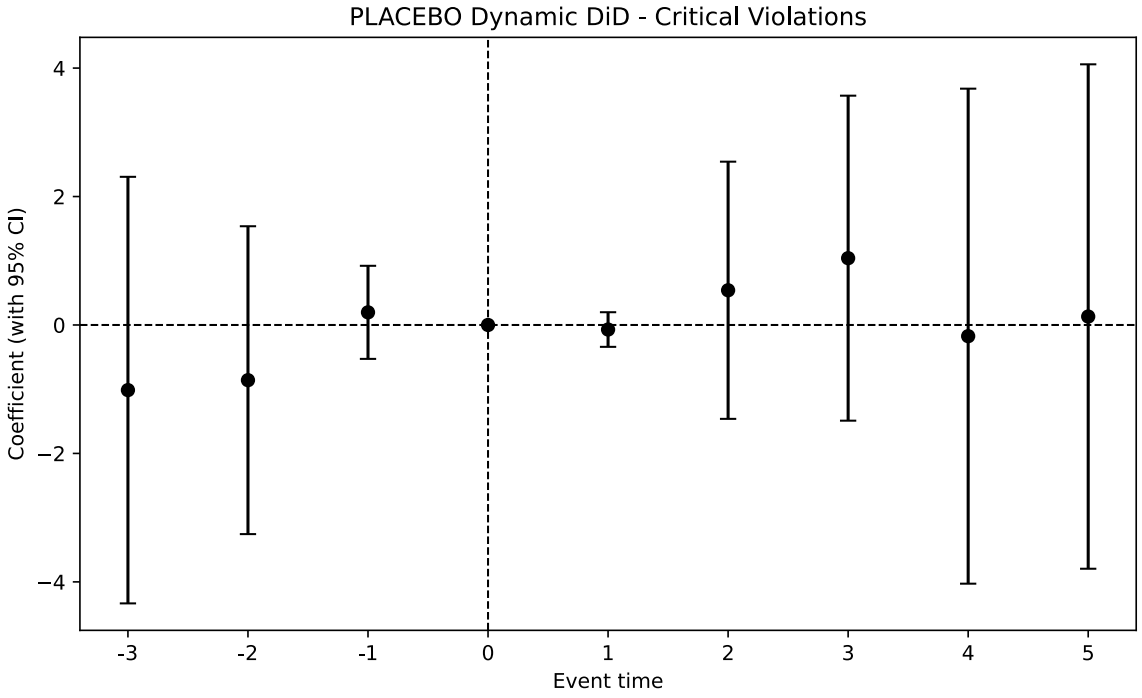


Figure 20: Stage-Specificity Difference-in-Differences Estimate — Critical Rule Violations per 1,000 Lines of Code

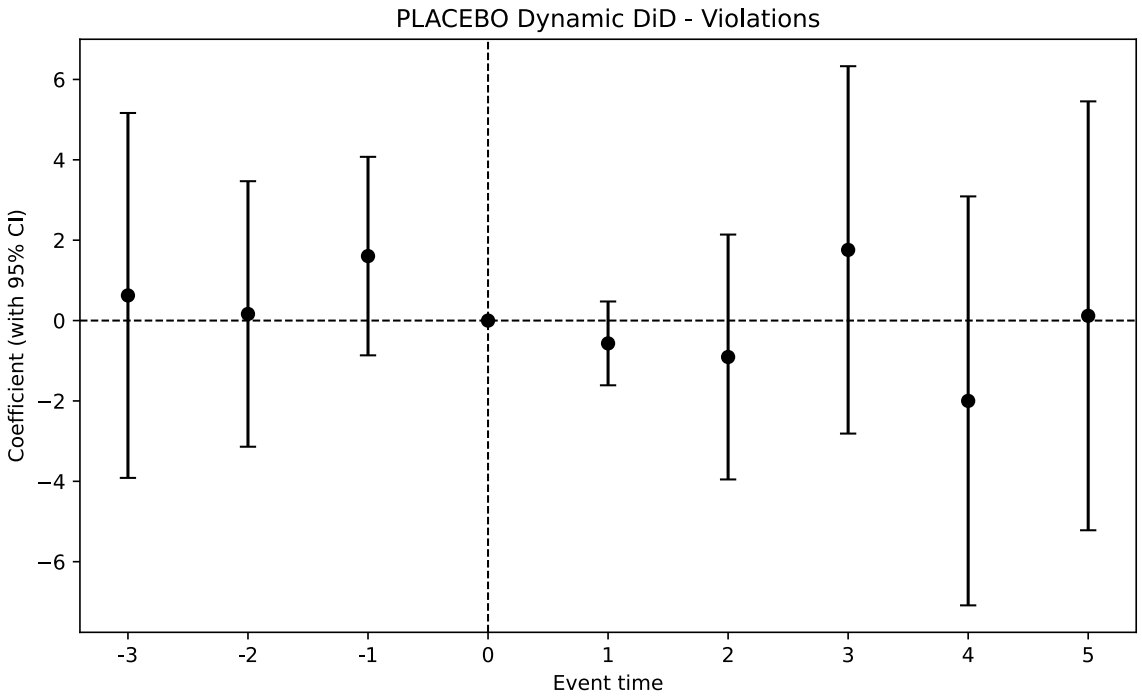


Figure 21: Stage-Specificity Difference-in-Differences Estimate — Rule Violations (All Severities) per 1,000 Lines of Code

Table 22: Stage-Specificity Difference-in-Differences Estimates for Critical Violations and All Violations

	Critical Violations			**Violations**		
	(1)	(2)	(3)	(4)	(5)	(6)
Treated × Post	0.652	0.037	1.999	-0.916	-0.813	-0.766
	(1.164)	(0.230)	(1.501)	(1.762)	(1.759)	(2.271)
Post × High Violations (=1)		-1.331**			-3.674**	
		(0.539)			(1.764)	
Treated × Post × High Violations (=1)		1.802			-0.374	
		(3.320)			(4.282)	
Post × Large Round			0.725			-0.117
			(0.652)			(1.419)
Treated × Post × Large Round			-3.662			-0.254
			(2.458)			(3.164)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	203,888	202,931	203,888	203,888	202,931	203,888
R^2	0.826	0.829	0.826	0.835	0.837	0.835

This table reports stage-specificity estimates from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Series B+ Treated × Post indicator. *Critical Violations* denote the highest-severity SonarQube rule violations per 1,000 lines of code, corresponding to issues with a significant probability of severe unintended consequences (e.g., production crashes or security flaws). *Violations (All)* denote total deviations from engineering best practices across all severity levels per 1,000 lines of code. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

Table 23: Stage-Specificity Difference-in-Differences Estimates for Maintainability Issues

	Maintainability Issues (All)			**Maintainability Issues (High)**			**Maintainability Issues (Low)**		
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Treated × Post	-1.098	-0.870	-1.060	0.775	0.151	2.133	-2.109	-0.730	-3.345
	(1.613)	(1.703)	(2.006)	(1.165)	(0.198)	(1.505)	(1.390)	(0.754)	(2.340)
Post × High Violations (=1)		-3.476**			-1.444***			-0.490	
		(1.687)			(0.526)			(0.540)	
Treated × Post × High Violations (=1)		-0.743			1.827			-4.110	
		(3.749)			(3.349)			(3.861)	
Post × Large Round			0.014			0.799			-0.852
			(1.391)			(0.642)			(0.541)
Treated × Post × Large Round			-0.099			-3.744			3.504
			(2.897)			(2.464)			(2.330)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	203,888	202,931	203,888	203,888	202,931	203,888	203,888	202,931	203,888
R^2	0.811	0.813	0.811	0.829	0.832	0.829	0.825	0.825	0.825

This table reports stage-specificity estimates from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Series B+ Treated × Post indicator. Maintainability Issues (All) capture total maintainability-related rule violations per 1,000 lines of code. High-Severity Maintainability Issues isolate the most severe maintainability problems, while Low-Severity Maintainability Issues capture more minor structural or readability deficiencies. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

Table 24: Stage-Specificity Difference-in-Differences Estimates for Security Hotspots

	Security Hotspots		
	(1)	(2)	(3)
Treated × Post	0.558**	0.444	0.889**
	(0.259)	(0.312)	(0.387)
Post × High Violations (=1)		0.223	
		(0.388)	
Treated × Post × High Violations (=1)		0.346	
		(0.552)	
Post × Large Round			0.349
			(0.429)
Treated × Post × Large Round			-1.030**
			(0.513)
Cohort × Firm FE	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes
# Obs	203,888	202,931	203,888
R^2	0.534	0.529	0.534

This table reports stage-specificity estimates from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Series B+ Treated × Post indicator. *Security Hotspots* measure security-sensitive code segments requiring manual audit per 1,000 lines of code. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

Table 25: Stage-Specificity Difference-in-Differences Estimates for Bugs and Reliability Issues

	Bugs			**Reliability Issues (High)**			**Reliability Issues (Low)**		
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Treated × Post	0.158	0.070	0.281	-0.108	-0.092	-0.113	-0.512	-0.675	-0.491
	(0.401)	(0.327)	(0.528)	(0.130)	(0.116)	(0.143)	(0.458)	(0.682)	(0.497)
Post × High Violations (=1)		-0.199			0.112			-0.165	
		(0.291)			(0.092)			(0.492)	
Treated × Post × High Violations (=1)		0.260			-0.046			0.482	
		(0.820)			(0.070)			(0.724)	
Post × Large Round			-0.162			-0.067			-0.414
			(0.323)			(0.072)			(0.305)
Treated × Post × Large Round			-0.157			0.061			0.270
			(0.623)			(0.088)			(0.415)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	203,888	202,931	203,888	203,888	202,931	203,888	203,888	202,931	203,888
R^2	0.826	0.826	0.826	0.758	0.758	0.758	0.815	0.816	0.816

This table reports stage-specificity estimates from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Series B+ Treated × Post indicator. *Bugs* capture coding errors that lead to runtime failures or unexpected behavior per 1,000 lines of code. *High-Severity Reliability Issues* represent defects posing significant risk to system stability, while *Low-Severity Reliability Issues* denote minor or edge-case flaws with limited immediate impact. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

Table 26: Stage-Specificity Difference-in-Differences Estimates for Technical Debt

	Technical Debt			**Technical Debt Ratio**		
	(1)	(2)	(3)	(4)	(5)	(6)
Treated × Post	-0.100	-0.146	-0.008	-0.013	-0.016	-0.003
	(0.317)	(0.288)	(0.395)	(0.032)	(0.030)	(0.039)
Post × High Violations (=1)		-0.528			-0.052	
		(0.394)			(0.040)	
Treated × Post × High Violations (=1)		0.126			0.008	
		(0.812)			(0.081)	
Post × Large Round			0.019			0.004
			(0.257)			(0.026)
Treated × Post × Large Round			-0.226			-0.025
			(0.618)			(0.062)
Cohort × Firm FE	Yes	Yes	Yes	Yes	Yes	Yes
Month × Cohort FE	Yes	Yes	Yes	Yes	Yes	Yes
# Obs	203,888	202,931	203,888	200,384	199,428	200,384
R^2	0.802	0.803	0.802	0.801	0.803	0.801

This table reports stage-specificity estimates from the specification $M_{i,c,t} = \beta VC_{i,c,t} + \chi_{c,i} + \delta_{c,t} + \gamma X_{i,t} + \varepsilon_{i,c,t}$, where $VC_{i,c,t}$ is the Series B+ Treated × Post indicator. *Technical Debt per Line of Code* measures the estimated remediation effort required to address code issues per unit of code, while the *Technical Debt Ratio* captures remediation effort relative to estimated development effort. See Appendix A, Table 16, for full variable definitions. *Note:* Standard errors in parentheses. * p<0.1, ** p<0.05, *** p<0.01. Standard errors are clustered by cohort and firm.

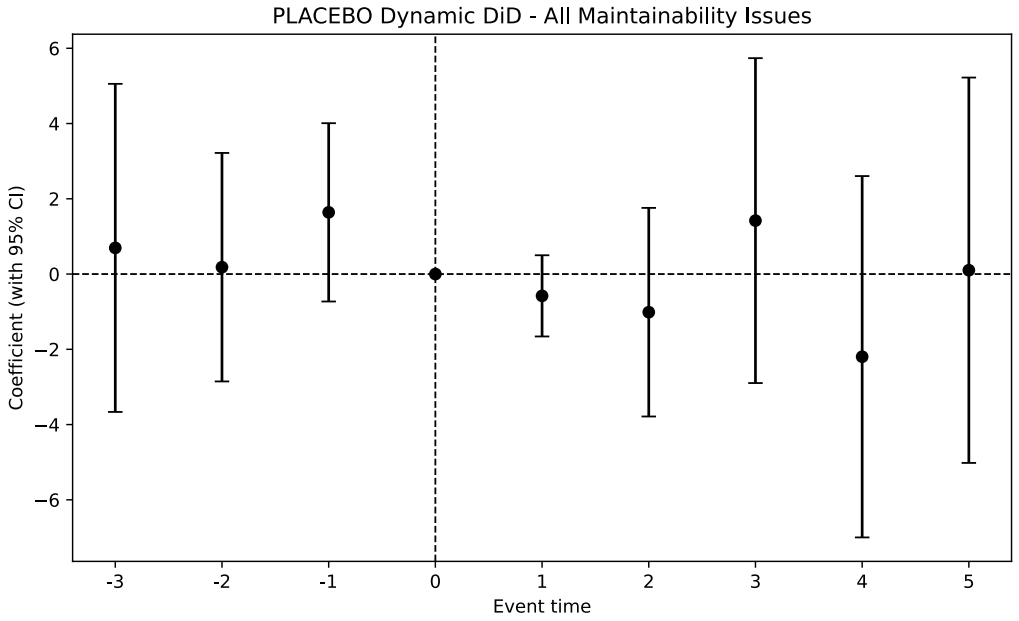


Figure 22: Stage-Specificity Difference-in-Differences Estimate — # of Maintainability Issues per 1,000 Lines of Code

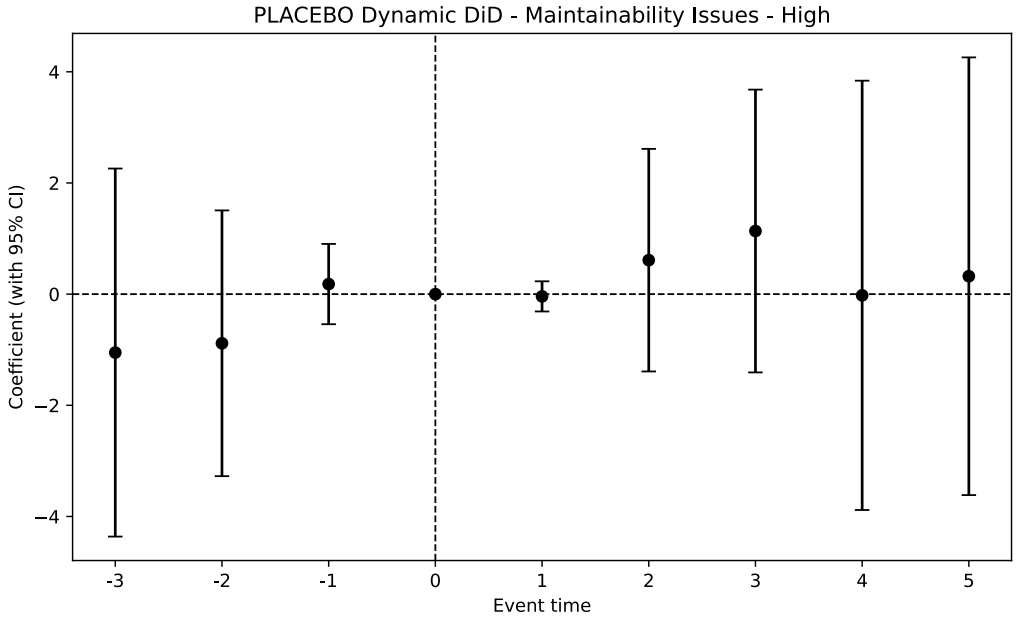


Figure 23: Stage-Specificity Difference-in-Differences Estimate — # of Maintainability Issues (High-Severity) per 1,000 Lines of Code

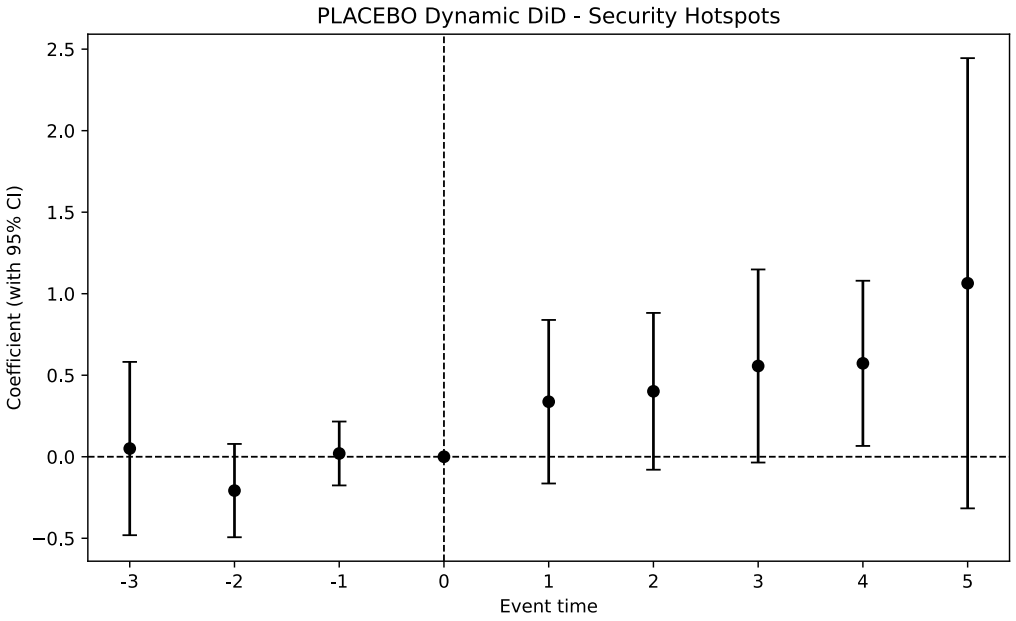
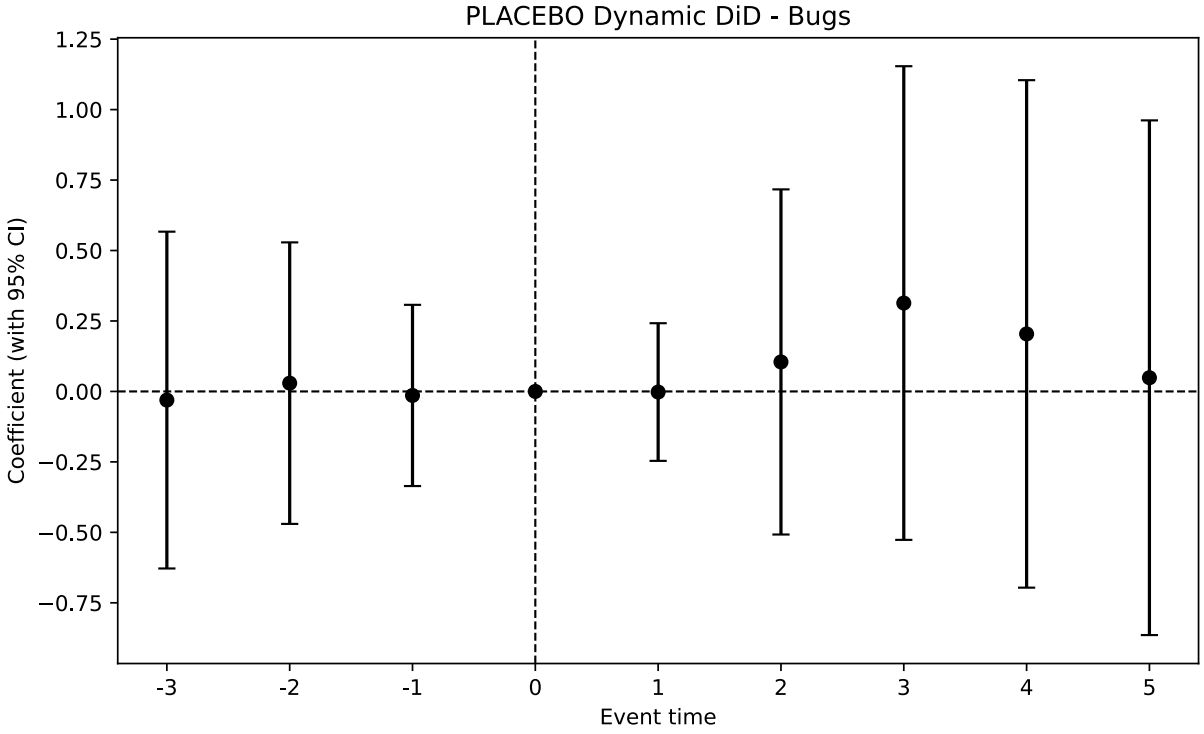
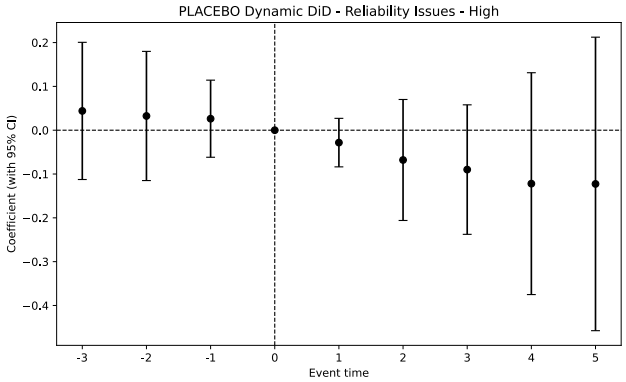


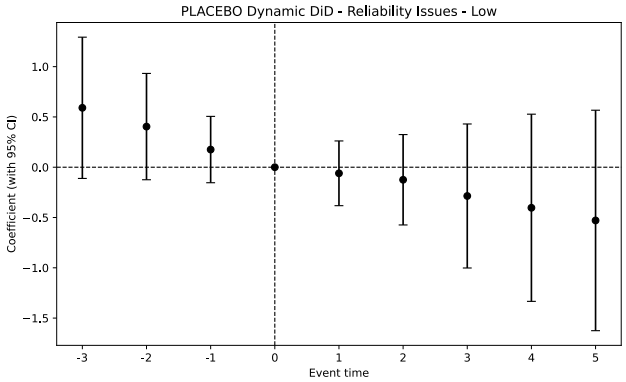
Figure 24: Stage-Specificity Difference-in-Differences Estimate — # of Security Hotspots per 1,000 Lines of Code



(a) Difference-in-Differences Estimate — Bugs per 1,000 lines of code

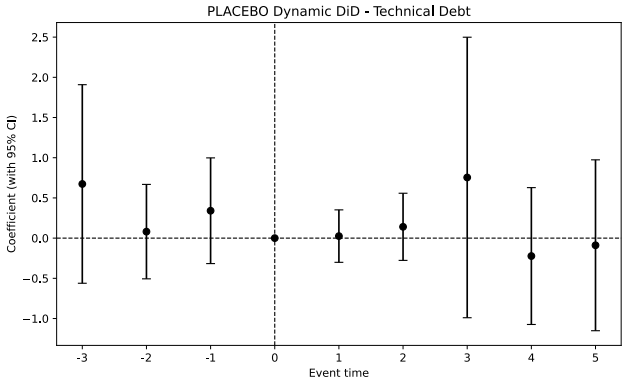


(b) Reliability Issues (High Severity) per 1,000 lines of code

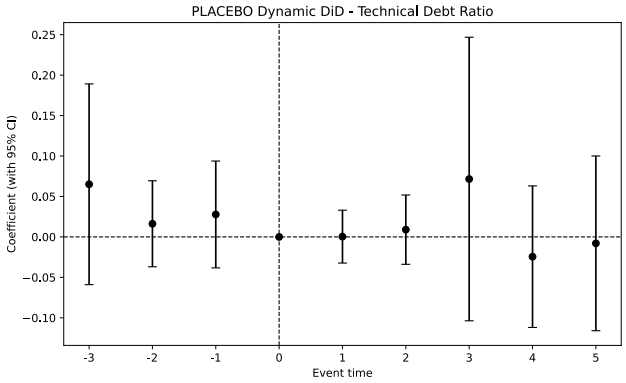


(c) Reliability Issues (Low Severity) per 1,000 lines of code

Figure 25: Stage-Specificity Dynamic Difference-in-Differences Estimates for Bugs and Reliability Issues



(a) Difference-in-Differences Estimate - $ of Technical Debt per line of code



(b) Difference-in-Differences Estimate - Ratio: Technical Debt over Software Assets

Figure 26: Stage-Specificity Difference-in-Differences Estimates for Technical Debt Measures