

Security technical debt and scanner-context artefacts in SME GitHub repositories: A paired pre/post-2023 SonarQube study

Tara Nikolić, Dario Ljubas, Goran Nikšić

University of Zagreb Faculty of organization and informatics

Department of information systems development

Pavlinska 2, 42000 Varaždin, Croatia

{tnikolic, dljubas}@foi.hr, gniksic21@student.foi.hr

Abstract. *This study examines how scanner scope affects temporal measurements of security technical debt in GitHub repositories maintained by small and medium-sized enterprises. A paired design compares repository states before and after 1 January 2023 using SonarQube, size-normalised debt measures, non-parametric tests, and manual classification of high-severity findings. Because AI use cannot be verified, the temporal comparison is descriptive rather than causal. Results indicate stable aggregate debt density and show that non-production files can substantially distort apparent severity changes. These findings demonstrate the importance of separating production from non-production code when comparing SAST results over time.*

Keywords. security technical debt, static analysis, SME, SonarQube, software security, AI-assisted development

1 Introduction

Temporal comparisons of static application security testing (SAST) output are increasingly employed to assess changes in software security risk. However, the validity of these comparisons depends on the scope defined by the scanner. Non-production artefacts such as test fixtures, localisation files, and vendored code often change more rapidly than production code and can disproportionately influence perceived shifts in severity. This study examines security technical debt in paired states of repositories maintained by small and medium-sized enterprises (SMEs) and investigates how the composition of production code affects the interpretation of longitudinal SAST results. The primary contribution is methodological: we evaluate what can be inferred from repository-level scanner output before attributing changes to developer behaviour or tooling.

The rapid adoption of large language model (LLM) coding assistants raises questions regarding potential changes in security debt over time. Nevertheless, this study neither observes nor infers AI usage, and no reliable ground truth for AI authorship is available in the public repositories analysed. The 1 January 2023 threshold serves solely to delineate two temporally distinct repository states, rather than to distinguish be-

tween exposed and unexposed groups. Consequently, all period results are descriptive rather than causal and cannot be used to estimate AI-induced security debt. Prior research on AI-assisted code is referenced as motivation, but it is not the construct measured in this study (Pearce et al., 2022; Sandoval et al., 2023).

Small and medium-sized enterprises (SMEs) are particularly vulnerable to this uncertainty. SMEs differ fundamentally from large organisations, operating within distinct economic and managerial contexts. They rarely maintain dedicated IT functions and often centralise decision-making in a single owner-manager who may lack the resources or expertise for systematic security governance (Devos et al., 2012). In contrast to large firms, which distribute verification across peer review, application-security teams, automated pipelines, and penetration testing, SMEs typically rely solely on the developer who accepted the suggestion (Chidukwani et al., 2022). If AI assistance accelerates the accumulation of security weaknesses, SMEs constitute the environment where this effect is likely to be most pronounced and least mitigated.

This paper makes three principal contributions. First, we present a paired, size-normalised assessment of security technical debt (STD) across 28 SME repositories, each analysed in two distinct states (56 analyses in total), encompassing 13.7 million lines of code. We detail the SonarQube configuration required to interpret the results and explain why the aggregate design is predisposed toward a null period difference. Second, manual classification of every high-severity finding reveals that the most prominent raw difference is attributable primarily to scanner context rather than to a genuine security signal. Third, we demonstrate that hotspot states observed in new research instances cannot be used to infer organisational triage behaviour and specify the additional evidence necessary to support such claims.

2 Background

This study is informed by two bodies of literature. The first conceptualizes security weaknesses as a form of technical debt, characterized by a principal, an interest rate, and a repayment mechanism. The second examines whether AI-assisted development alters the rate at which such debt accumulates. Evidence regarding this latter question remains inconclusive.

2.1 From technical debt to security debt

The technical debt (TD) metaphor conceptualizes expedient implementation choices as a loan, with interest repaid through increased maintenance costs (Avgeriou et al., 2016; Cunningham, 1993; Li et al., 2015). However, security does not fully correspond to this framework. An unresolved vulnerability remains dormant until exploited, and the resulting cost may not correspond to the original shortcut. Rindell et al. (2019) address this asymmetry by way of the ACT-S framework, which reinterprets security risk as technical debt and enables established management practices such as identification, measurement, prioritisation, and repayment. Martinez et al. (2021) define security debt as the incomplete or incorrect set of security properties required by a system in its operational context, with interest measured in accumulated risk rather than maintenance effort.

AI-induced security debt is defined in the literature as accumulated and unresolved security risk that arises or is amplified when software artefacts are produced or influenced by AI tools. Rather than constituting a new category, it represents an accelerated source of the previously described debt, altering three key properties: the principal accumulates more rapidly due to code generation at machine speed, the interest is more deeply latent, as debt is often incurred by individuals unable to recognize it, and ownership becomes diffuse, since the originating author is a probabilistic model. Four mechanisms are identified as compounding factors: insecure generation under default conditions (Pearce et al., 2022; Veracode, 2025), human overconfidence, where developers using AI assistants produce less secure code while perceiving it as more secure (Perry et al., 2023), a mismatch between code generation and review rates (DeBellis et al., 2024), and the introduction of new attack surfaces in LLM-integrated systems, such as prompt-to-SQL injections demonstrated by Pedro et al. (2025). From an architectural perspective, Recupito et al. (2024) identify Undeclared Consumers-components that silently use model outputs without access control-as having the highest security impact among the AI-specific anti-patterns examined (Sculley et al., 2015).

2.2 The verification gap

Functionally correct code does not guarantee security, however, the evidence regarding this relationship is mixed, and measurement methodologies vary. Scenario-based prompts reveal high vulnerability rates (Pearce et al., 2022). Conversational models may identify issues when prompted but often still produce non-robust default outputs (Khoury et al., 2023). Repository mining has identified weaknesses across 43 CWE classes in Copilot-generated code that has already been merged into GitHub projects (Fu et al., 2025). Controlled trials, however, do not consistently show an increase in vulnerabilities (Sandoval et al., 2023). Therefore, it is defensible to conclude that AI-generated code is not inherently less secure, but its provenance does not guarantee security, and all generated code should be subject to the same controls as any other.

For small and medium-sized enterprises (SMEs), the critical gap lies between the level of verification required and the available oversight capacity. Security reviews are often conducted by the same developer who accepted the generated code, if they occur at all (Chidukwani et al., 2022). Web applications amplify this risk because customer-facing systems combine commercial significance with direct Internet exposure. Additionally, reliance on frameworks, cloud services, and external contractors obscures responsibility for secure defaults, which may not be present (Devos et al., 2012).

Static application security testing (SAST) represents the most practical initial control in this context. SAST operates on every code change and identifies known source-to-sink patterns, dangerous application programming interfaces (APIs), and hard-coded secrets (GitHub, 2026; SonarSource, 2026a). However, SAST cannot evaluate business-logic authorization or deployment context, and it generates both false positives and false negatives (Aloraini et al., 2019; Charoenwet et al., 2024). Organisational response requires separate evidence from existing review systems or workflows, a scanner run created for research measures detections, not whether an organisation is classifying them elsewhere for analysis.

3 Methodology

Measuring security debt across two time points requires a research design that avoids confounding comparisons with differences between projects. In this study, each repository is analyzed twice, once in a pre-threshold state and once in a post-threshold state, making sure that language, domain, and team remain constant. However, this approach results in nested rather than independent states, which constrains the interpretation of results discussed in Section 4.

3.1 Research design

An empirical quantitative design is employed to examine security technical debt in GitHub repositories owned by small and medium-sized enterprises (SMEs). Measurement is based on SonarQube, which reports security findings and estimates remediation effort using a rule-based quality model (Letouzey & Ilkiewicz, 2012; SonarSource, 2026b). The estimated remediation effort serves as a relative indicator of security technical debt (STD) (Li et al., 2015).

The date 1 January 2023 serves as an operational threshold distinguishing the period before the widespread availability of code assistants from the period after, based on Copilot's general availability in mid-2022 (Pearce et al., 2022) and ChatGPT's public release in late 2022. This threshold functions as a coarse research proxy rather than a definitive treatment indicator. Pre-threshold repositories may already include assisted code, while post-threshold repositories may contain none; thus, misclassification is anticipated in both directions, which biases any period comparison toward a null result. Consequently, all findings are interpreted as associations, and the period comparison is presented as a secondary, exploratory observation.

The study employs a paired design: each selected repository is analyzed twice, once in a pre-threshold state and once in a post-threshold state, reconstructed from its commit history. Each project thus serves as its own control, eliminating between-project variance in language, domain, team composition, and coding conventions from the period comparison. This design necessitates the use of paired statistical tests throughout the analysis.

The two states of each repository are not independent samples; the post-threshold snapshot is a superset of the pre-threshold state. Consequently, any density calculated on the post-threshold state represents an average over a code base in which post-threshold code comprises only a fraction of the total, potentially biasing the period comparison toward a null result regardless of actual changes in code quality. The median repository increased in size from 122.2 to 178.9 KLOC, indicating that post-threshold code accounts for approximately one third of the later snapshot and is weighted accordingly. The aggregate is reported because it reflects the security state that an organisation actually maintains. However, a flat aggregate density does not demonstrate that recently written code is comparable in quality to older code. Commit-level attribution would directly isolate post-threshold code; the lack of such attribution represents the principal design limitation of this study.

This study addresses three research questions:

- **RQ1:** What is the level and density of STD in SME-related GitHub repositories?
- **RQ2:** Which vulnerability categories contribute most to STD?

- **RQ3:** Is there a relationship between project size, project complexity, and STD?

3.2 Repository selection and data preparation

The dataset consists of publicly available GitHub repositories owned by SMEs or SME-like organisations. Inclusion criteria required repositories to be public, contain sufficient project-specific code in a supported language, be compatible with SonarQube, have identifiable ownership, and be assignable to a specific period. Exclusion criteria comprised forks without substantial development, tutorial or demo repositories, archived or inactive projects, repositories mainly comprising third-party libraries, generated or boilerplate-only code, unclear SME connections, and those that could not be analysed for technical reasons.

A repository was classified as SME-related if the owning organisation satisfied at least two proxy criteria: (1) the public profile or linked website describes a commercial product or service, (2) the number of public contributors does not exceed 50, (3) cross-referencing via LinkedIn or a company website indicates fewer than 250 employees, consistent with the EU SME definition (European Commission, 2003), or (4) there are no clear indicators of enterprise-scale activity. These criteria serve as proxy signals rather than verified records. A single coder conducted the classification process according to set decision rules. Ambiguous cases were excluded rather than resolved by subjective judgment. Inter-rater agreement was not evaluated.

A target sample of approximately 60 repositories per group was established, consistent with non-parametric methods commonly used in empirical software engineering (Wohlin et al., 2012). The final sample size was smaller due to the combined requirements of pairing, tool compatibility, and successful analysis; the realised figures are reported in Section 4.1. Duplicate and near-duplicate repositories were removed. Where technically feasible, third-party dependencies, vendor folders, generated and minified files, and framework boilerplate were excluded to ensure the analysis concentrated on project-specific source code.

3.3 Metrics

The primary dependent variable is STD (Li et al., 2015), which is measured as the estimated security remediation effort reported by SonarQube according to its metric definition (Letouzey & Ilkiewicz, 2012; SonarSource, 2026b). Remediation effort is considered a consistent relative metric rather than an exact measure of developer time. To account for the mechanical accumulation of findings in larger repositories, two normalised metrics are also defined:

$$\text{STD Density} = \frac{\text{Security Remediation Effort (min)}}{\text{KLOC}} \quad (1)$$

$$\text{Vulnerability Density} = \frac{\text{Detected Vulnerabilities}}{\text{KLOC}} \quad (2)$$

KLOC refers to thousands of logical lines of code, as measured by SonarQube's `ncloc` metric. Both raw and normalised metrics are reported, as they rank projects differently and address distinct questions. Raw effort quantifies the remediation burden faced by an organisation, whereas density reflects the security quality of the code regardless of its size.

SonarQube assigns a fixed remediation cost to each rule; therefore, the total remediation effort is calculated as a rule-weighted function of the number of findings, rather than as an independently observed metric. As a result, STD density and vulnerability density are inherently related by design and should be interpreted as alternative representations of a single underlying quantity, rather than as independent confirmations of one another.

In addition to the previously defined security metrics, each analysis collects project size (measured in logical lines of code) and five complexity measures: files, functions, classes, cyclomatic complexity, and cognitive complexity. These serve as independent variables for RQ3. Security hotspots and the SonarQube A-E security-review rating characterise the state of each research instance, however, they are not considered indicators of organisational process. The main programming language and period assignment are recorded as descriptive variables, with the programming language also treated as a potential confounder. To ensure analytical transparency, the SonarQube version, the quality profiles for each activated language plugin, the retrieved metric keys, and the fetch date are documented for each analysis.

3.4 Statistical analysis

The analysis commences with descriptive statistics, including repository size, language distribution, vulnerability counts and density, total STD, and STD density. Language is reported descriptively because vulnerability profiles differ across languages (Croft et al., 2022). The distributions show strong right skewness, which motivates the use of non-parametric methods throughout. Shapiro-Wilk test results are provided for completeness but are not emphasized, as the test is underpowered at $n = 28$.

Spearman's rank correlation is used to examine the relationships between size, complexity, and STD (RQ3). Given the paired design, the Wilcoxon signed-rank test is applied for period comparisons, while the unpaired Mann-Whitney U test is reported as a robustness check. The Kruskal-Wallis test is employed when

more than two groups are compared, as in the language analysis. The significance level is set at $\alpha = 0.05$. Effect sizes are reported as the matched-pairs rank-biserial correlation r where applicable. For monotonically increasing quantities such as code size, a rank-biserial value near unity indicates that nearly every repository changed in the same direction, which is expected for cumulative code bases and should not be interpreted as a measure of practical significance.

Given the exploratory nature of the study and the presence of multiple comparisons, p -values in the time frame analysis are adjusted using the Benjamini-Hochberg procedure (denoted p_{BH}) and interpreted with caution. The correction family includes the period comparisons reported in Section 4.5: size, absolute remediation effort, vulnerability count, STD density, and vulnerability density. Differences in the distribution of severity levels between periods are assessed using the χ^2 test of independence.

The Spearman correlations for RQ3 are based on 56 analyses from 28 repositories. The two observations from each repository are not independent, as they share language, domain, team, and most of their code. Consequently, correlations computed on the pooled set have effective degrees of freedom below the nominal n , resulting in anti-conservative p -values. The coefficients reported in Section 4.4 represent pooled descriptive values. Their p -values should be interpreted with caution, as independent inferential tests would require a one-state-per-repository or mixed-effects specification.

3.5 Manual inspection protocol

Given that SAST output frequently contains context-dependent false positives (Aloraini et al., 2019; Charoenwet et al., 2024), all findings at the two highest severity levels (CRITICAL and BLOCKER) are manually classified rather than accepted without scrutiny. Each finding is categorized as either production or non-production code based on its file path. Non-production code includes test directories, localisation and translation resources, vendored third-party code, fixtures, and mocks. Severity analysis is reported twice: once for all findings and once restricted to production code. This approach explicitly demonstrates the influence of scanner context on newly observed effects. Rules generating large clusters of findings are examined individually and traced to their source repositories. The assignment rule is strictly mechanical and was applied by a single coder: no inter-rater agreement statistic is reported. Because the rule depends solely on directory names and requires no subjective judgment regarding the findings, the risk of misclassification is considered low, although this has not been empirically verified.

3.6 Validity and limitations

Construct validity. This study cannot confirm whether specific code was AI-generated. The 2023 cut-off serves only as a coarse temporal proxy rather than a treatment indicator, and period assignment relies on commit history. The central construct identified in the motivating literature, namely AI-induced security debt, is not directly measured in this analysis; therefore, none of the subsequent results should be interpreted as estimates of this construct. Static Application Security Testing (SAST) tools are known to produce both false positives and false negatives (Alo-raini et al., 2019; Charoenwet et al., 2024). To address this, the SonarQube configuration is fully reported, and all high-severity findings are manually inspected as described in Section 3.5. The exclusive dependence on a single tool constitutes an additional limitation (Charoenwet et al., 2024), but this approach was adopted to maintain analytical consistency and guarantee a comparable remediation-effort metric across all repositories.

Internal validity. The nested-snapshot problem described in Section 3.1 represents the primary threat to internal validity. The programming language functions as a potential confounder, as vulnerability profiles vary across languages (Croft et al., 2022). Language is recorded, reported descriptively, and its effect on both normalised metrics is tested; however, it is not statistically controlled for in the main models, which is acknowledged as a limitation. Additional factors, including developer experience, security practice maturity, and repository activity, may also influence the results and are incorporated into the interpretation of findings. Estimates obtained from 28 matched pairs are imprecise; therefore, a small true difference remains plausible given the data. The period comparison backs the conclusion that no aggregate change was detected, rather than providing evidence of equivalence.

External validity. Public repositories may not accurately represent private SME projects, and SME classification is based on public signals rather than verified data. The sample is mainly composed of PHP projects, with PHP representing 31 out of 56 analyses. The language inclusion criterion was revised during data collection to accommodate this. Consequently, the findings primarily reflect characteristics of SME-maintained PHP web projects rather than SME software development as a whole. Additionally, the Kruskal-Wallis language tests reported in Section 4.3 lack sufficient statistical power to exclude programming language as a confounding factor. Therefore, extrapolation beyond this specific population is not justified.

Reliability. Remediation effort is estimated and utilized as a relative metric. The selection criteria, exclusion logic, key SonarQube configuration, and analysis decisions are reported to enhance transparency. Analysis scripts, repository lists, and complete profile exports are retained internally. As no public replication package is provided and profile version identifiers were not recorded, the study does not assert independent replicability.

4 Results

The results are structured to ensure that the analysis configuration is clearly established prior to any comparative assessment. Aggregate quantities are presented initially, followed by manual inspection, as the raw data and the inspected results may lead to different interpretations. Sections 4.2, 4.3, and 4.4 address RQ1, RQ2, and RQ3, respectively, while Sections 4.5 and 4.6 present the exploratory period comparison and the manual inspection of high-severity findings.

4.1 Configuration and realised sample

All 56 analyses were conducted on a single SonarQube instance (v26.6.0.123539, metrics retrieved 22 July 2026) using the built-in Sonar way profiles across all 27 activated language plugins, including the dedicated secrets profile. Profiles and metric keys were identical for every analysis, so ruleset drift cannot account for any difference reported below. The profile name and server version were recorded, but profile version identifiers were not. Because the contents of the Sonar way profiles change across SonarQube releases, exact replication would require the profile definitions shipped with v26.6.0 rather than the profile name alone.

After applying the criteria in Section 3.2, 28 repositories remained, each analysed in a pre- and post-threshold state, resulting in 28 matched pairs. The realised sample is not language-balanced, and this is its principal external-validity constraint. PHP constitutes the majority (31 of 56 analyses), followed by TypeScript (7), JavaScript (6), Python (5), CSS (3), Java (2), and XML (2); no C# repository met the remaining criteria. The language criterion stated in Section 3.2 was revised during data collection. PHP was included after it became evident that the originally listed languages would not provide a sufficient number of SME-owned repositories that met the pairing and tool-compatibility requirements, and no C# repository qualified. Consequently, the final language composition shows repository availability rather than initial study design. Language is further examined as a potential confounder in Section 4.3.

Table 1. Descriptive statistics by period ($n = 28$ per group). Medians with totals in parentheses; densities are ratios and have no total.

	Pre-2023	Post-2023
Size (KLOC)	122.2 (5,267)	178.9 (8,451)
Effort (min)	267.5 (10,551)	275.5 (26,982)
Vulnerabilities	8.5 (577)	10.0 (906)
STD density (min/KLOC)	1.617	1.619
Vuln. density (/KLOC)	0.065	0.062

4.2 RQ1: Level and density of STD

Across all 56 analyses, SonarQube identified 1,483 vulnerabilities and 6,803 security hotspots, corresponding to an estimated 37,533 minutes of remediation effort over 13.7 million lines of code (Table 1). All variables show strong right skewness, and the Shapiro-Wilk test rejected normality in all cases ($W = 0.28\text{--}0.72$, $p < 0.001$), supporting the use of non-parametric tests.

The median repository requires 272 minutes of security remediation effort, corresponding to 1.62 minutes per thousand lines of code (min/KLOC) and 0.06 vulnerabilities per KLOC. In absolute terms, this represents approximately four and a half hours of estimated work for a typical project. However, security debt is highly concentrated. Five repositories require no security remediation effort, while the two largest debtors account for 36.7% of the total. Raw effort and density yield different project rankings, which is why both metrics are reported. For example, the densest repository in the sample (90.7 min/KLOC) is a 717-line project with only 65 minutes of debt, making it undetectable in a raw-effort ranking.

This concentration imposes a constraint on all aggregate statistics presented in the paper. When one third of the total debt resides in two of 28 repositories, medians, totals, and pooled correlations may be disproportionately influenced by a small number of high-leverage projects. Consequently, the correlations reported in Section 4.4 should be interpreted descriptively.

Among the 56 analyses, 54 received the lowest security-review rating. The remaining two analyses contained no hotspots and were assigned an A rating by default. All 6,803 hotspots were marked as TO_REVIEW on the SonarQube instances created for this study.

These states describe the research instances and must not be interpreted as evidence of organisational review behaviour. Detection and disposition were not measured symmetrically: detections were actively generated by the researchers by scanning repositories on newly created SonarQube instances, whereas dispositions were only passively observed on those same instances. Detection was therefore expected for repositories of this size, while an existing disposition could not appear unless prior review history had been imported or a new review had been performed after the

Table 2. Rules ranked by contribution to STD (all 56 analyses).

Rule	n	Effort	Share
S5122 Permissive CORS	51	12,240	32.6%
S5443 Temp file, writable dir	260	7,800	20.8%
S2068 Hard-coded credentials	149	4,470	11.9%
S6418 Hard-coded secrets	98	2,940	7.8%
S5725 Missing integrity check	509	2,545	6.8%
S6865 Service account perms	23	1,380	3.7%
S6268 Sanitization disabled	44	1,320	3.5%
S6358 Android backup unrestricted	8	720	1.9%
All other rules	341	4,118	11.0%
Total	1,483	37,533	100%

research scan. Organisations may review equivalent findings in another SonarQube deployment, an issue tracker, a pull-request workflow, or another security tool. Consequently, the observed TO_REVIEW states neither demonstrate an absence of organisational triage nor permit estimation of an organisational triage rate. No empirical detection-versus-disposition comparison is therefore reported.

4.3 RQ2: Categories contributing most to STD

The findings are concentrated within a small set of rules, though their distribution changes depending on whether they are ranked by count or remediation effort. By count, missing integrity checks on remote artefacts (S5725) are most common, with 509 findings, followed by temporary files in publicly writable directories (S5443, 260), hard-coded credentials (S2068, 149), and hard-coded secrets (S6418, 98). Together, these four rules represent 68.5% of the 1,483 total vulnerability findings. Since RQ2 addresses contribution to security technical debt (STD) rather than finding counts, Table 2 presents rules ranked by remediation effort. This results in a substantially different ordering. S5725, despite being the most frequent rule, contributes only 6.8% of total remediation effort because SonarQube assigns it five minutes per finding. In contrast, permissive CORS configuration (S5122) accounts for only 51 findings, but each requires four hours to address. This issue represents 32.6% of the total measured debt, making it the largest single contributor in the sample. Ranking by count would obscure this impact. Therefore, a remediation queue ordered by finding count versus one ordered by estimated effort would direct a subject matter expert's attention to almost entirely different sets of rules.

Among identified hotspots, weak cryptography is the most prevalent, with (1,925) findings. This is followed by denial of service (1,348), a residual 'others' category (1,326), unencrypted data (999), authentication (511), remote code execution (330), and permissions

(217). Hotspots do not include remediation estimates and are therefore reported solely by count. Fig. 1(a) presents the distribution by period.

Language did not confound the results. Among the four languages with at least four analyses (PHP, TypeScript, JavaScript, Python), the Kruskal-Wallis test found no significant difference in STD density ($H = 0.95$, $p = 0.812$) or vulnerability density ($H = 0.35$, $p = 0.950$). However, these tests are underpowered, and a null result provides only limited evidence. Therefore, while the dominance of PHP in the sample does not solely account for the reported density figures, it cannot be concluded that programming language is irrelevant.

4.4 RQ3: Size, complexity and STD

The result depends on the specific construct of security technical debt (STD) under consideration, as the two constructs produce divergent results.

Larger and more complex projects exhibit substantially greater absolute security technical debt. Project size demonstrates a moderate correlation with remediation effort ($\rho = 0.45$) and vulnerability count ($\rho = 0.49$), both statistically significant ($p < 0.001$, see Fig. 1(b)). All complexity indicators show notable correlations with both outcomes: number of files ($\rho = 0.45 / 0.44$), classes ($\rho = 0.42 / 0.45$), functions ($\rho = 0.37 / 0.42$), cyclomatic complexity ($\rho = 0.33 / 0.41$), and cognitive complexity ($\rho = 0.31 / 0.44$), all $p < 0.05$. These coefficients are pooled and should be interpreted in light of the non-independence discussed in Section 3.4 and the leverage check described in Section 4.2.

After normalisation, the correlations become small, negative, and statistically non-significant: project size versus STD density yields $\rho = -0.16$ ($p = 0.233$), and versus vulnerability density $\rho = -0.14$ ($p = 0.321$). This result indicates a lack of evidence for a positive relationship between size and density, rather than a reversal; two non-significant coefficients of this magnitude do not demonstrate that density decreases with size. The data support the more cautious conclusion that, in this sample, security debt accumulates approximately in proportion to code volume. Furthermore, complexity indicators do not outperform raw size as predictors, suggesting that the total amount of code, rather than its complexity, primarily determines the extent of security debt in these repositories.

4.5 Exploratory period comparison

The comparison presented is secondary and does not imply causality (Section 3.1). The size of the repository increased substantially, with the median repository gaining approximately 46% in lines of code ($r = +0.91$, $p_{BH} < 0.001$), accompanied by parallel growth in files, functions, and complexity. As

noted in Section 3.4, near-universal growth is expected for cumulative code bases; therefore, the effect size should not be interpreted as substantively meaningful. In contrast, the security metrics do not exhibit similar trends. Absolute remediation effort ($r = +0.24$, $p_{BH} = 0.343$) and vulnerability count ($r = +0.09$, $p_{BH} = 0.683$) increased only in the median and not significantly, while all normalized metrics remained stable: STD density was essentially unchanged (1.617 vs. 1.619 min/KLOC, $p_{BH} = 0.523$), and vulnerability density was marginally lower ($r = -0.30$, $p_{BH} = 0.255$). The unpaired Mann-Whitney test verified these findings throughout ($p = 0.63$ -0.99). Fig. 2 presents the paired trajectories, where the size panel diverges sharply, while both density panels remain level.

4.6 Manual inspection of high-severity findings

The inspection protocol described in Section 3.5 substantially alters the interpretation of the measured period difference.

The proportion of blocker and critical vulnerabilities increases from 13.3% to 47.5% ($\chi^2(3) = 206.5$, $p < 0.001$), representing the largest period difference in the dataset and, if interpreted without further analysis, appearing to strongly support the acceleration hypothesis. However, classification of all 507 high-severity findings by file path indicates that this increase is primarily an artefact: 15.6% of pre-threshold high-severity findings are located outside production code, compared to 78.4% post-threshold. Two sources account for the majority of these findings. Rule S5443 (temporary file in a world-writable directory) generated 260 findings, with 257 located in `__tests__` directories and 257 originating from a single repository, primarily flagging scratch files written to `/tmp` in unit tests. Rule S6418 generated 56 findings in translation files, triggered by words such as `token` and `secret` used as user-interface labels.

When analysis is restricted to production code, the observed shift decreases to 12.2% versus 17.3% (Fig. 3). Although this difference remains detectable ($\chi^2(3) = 22.1$, $p < 0.001$), it represents only about one quarter of the initial apparent magnitude. The difference is no longer statistically significant when tested per repository ($p = 0.079$), and it disappears entirely when the single dominant repository is excluded ($p = 0.143$). The χ^2 test treats findings as independent observations, despite clustering within repositories and rules, which inflates its significance. The per-repository test is therefore more reliable, and it does not reach significance. Some genuine production findings persist, including hard-coded credentials, unverified TLS certificates, and unauthenticated endpoints, indicating that the residual effect is not zero. However, the primary conclusion is that the observed severity shift is mainly a scanner-context artefact. This anal-

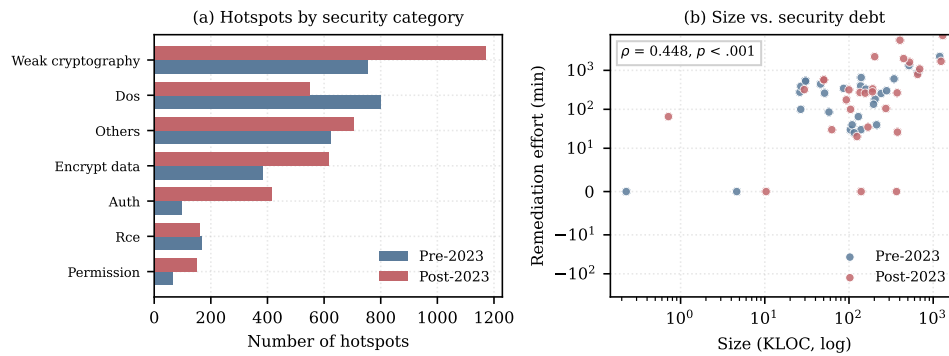


Figure 1. (a) Security hotspots by category and period. (b) Repository size versus estimated remediation effort, with pooled Spearman's ρ .

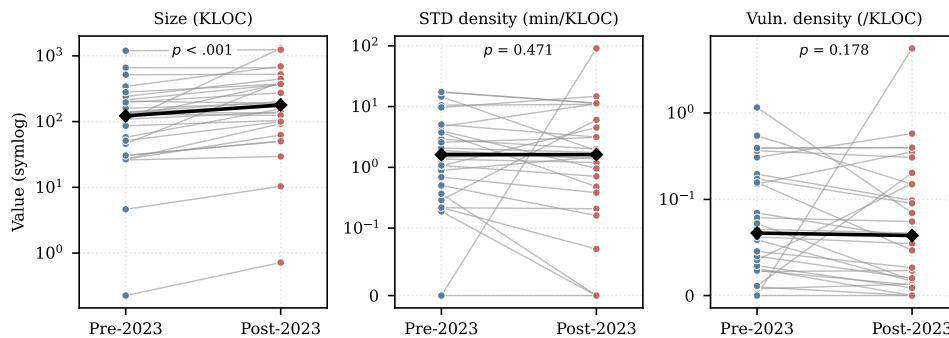


Figure 2. Paired repository trajectories (symmetric-log ordinate). Grey lines connect the two states of each repository; the black line joins group medians.

ysis also illustrates the practical cost of unfiltered scanner output: a subject matter expert acting on the raw findings would have allocated most of the remediation budget to test fixtures and translation strings.

5 Discussion

Taken together, these findings redirect attention from evaluating whether the assistant era has increased the density of security weaknesses to examining how scanner scope affects measurements of technical debt in expanding code bases. The period comparison supports only a limited descriptive claim, whereas the scanner-context finding supports a broader methodological conclusion. The hotspot states observed on the newly created research instances do not support conclusions about organisational triage behaviour.

5.1 Aggregate debt density did not change, the code base grew

The initial hypothesis that development in the assistant era would accelerate the accumulation of security debt is not supported by the data in this sample. Although repositories increased substantially in size and their absolute security debt grew proportionally, the density measures remained stable. An STD density of 1.617

compared to 1.619 min/KLOC represents a result that is effectively null for this type of measurement.

This null result should be considered in the context of the dilution described in Section 3.1. Degradation restricted to new code would appear as only a minor aggregate change. Therefore, the most defensible interpretation is more limited than a general null for AI-assisted code: the density of these repositories, as maintained artefacts, did not change across the threshold, even though their size increased by nearly fifty percent. Three explanations remain consistent with the data: assistance may not degrade shipped code in these contexts, the threshold may be too coarse to detect changes, or any effect in new code may be obscured by aggregation. Therefore, this result should be considered provisional.

This result does not contradict the scenario-based literature, but rather measures a different aspect. While Pearce et al. (2022) and Veracode (2025) assess model outputs in isolation, the present measurement captures the aggregate state of a repository following organisational review, testing, and iteration, with results affected by historical code. The finding corresponds more closely to Sandoval et al. (2023), who also observed no significant increase under controlled conditions, although their study design identifies a treatment, whereas this study does not.

5.2 The artefact problem is the practical finding

The severity analysis presented in Section 4.6 represents the most consequential finding for practitioners and is the least affected by the aforementioned design limitations. A 34-percentage-point increase in high-severity findings could be interpreted as evidence of an AI-driven security crisis. However, manual inspection reduces this effect to a residual level that is not statistically significant when assessed at the repository level and disappears entirely when a single repository is excluded. This phenomenon happens because expanding projects tend to add test suites and localisation files at a faster rate than production code, leading scanners to flag patterns in these directories that are benign in context but would be serious if found elsewhere.

This evidence has methodological implications that go beyond the scope of this study. Temporal comparisons of static application security testing (SAST) output that do not account for the composition of production versus non-production code risk conflating test-suite expansion with actual security degradation. Future research on AI and code security should routinely report severity distributions limited to production code, conduct analyses at the repository level rather than the individual finding level, and examine the rules underlying any substantial cluster of findings prior to drawing conclusions.

5.3 Implications for SME governance

The primary practical implication supported by this study is the need for context-aware interpretation of scanner output. Most post-threshold high-severity findings were located outside production code. Organisations that use raw scanner counts as a remediation queue risk misallocating limited resources. Distinguishing test, vendor, fixture, and localisation paths from production-security reporting can enhance signal quality without fully suppressing these findings.

Remediation priorities should reflect both exposure and estimated remediation effort. Absolute measured

security debt was associated with code volume and concentrated in a small subset of repositories. Prioritisation should therefore consider customer exposure and remediation effort rather than distribute attention uniformly or rank rules solely by finding count. Generated code, when present, should be subject to the same controls as other contributions because provenance alone does not guarantee security.

The hotspot results support recommendations for future measurement rather than conclusions about SME governance. A study of organisational disposition would require observation of organisations' existing scanner deployments or triangulation of scanner states with issue trackers, pull requests, security tickets, and interviews. Detection and disposition would need to be defined and observed within the same observation window. The newly created SonarQube instances used in this study cannot reveal whether, where, or how the sampled organisations review security findings.

6 Conclusion and future work

Analysis of 28 SME repositories in matched pre- and post-2023 states indicates that aggregate size-normalised security debt remained unchanged. Code bases expanded by a median of 46%, yet both security technical debt (STD) density and vulnerability density were stable. Absolute security debt correlated with code volume rather than code complexity. The identified vulnerabilities primarily reflected common web weaknesses, rather than issues unique to generated code.

The only substantial period difference in the raw data, a rise in high-severity findings from 13.3% to 47.5%, was determined through manual inspection of all 507 of those findings to be primarily a scanner-context artefact, driven by test and localisation directories. This difference disappears when assessed at the repository level or when a single repository is excluded. Therefore, period comparisons of static application security testing (SAST) output that do not control for production-code composition are unreliable.

Subsequent research should directly address the principal limitations of this study. Attributing findings to their introducing commit would replace the diluted aggregate with a measurement focused on newly written code, representing the most impactful improvement to the study design. Repositories with verified labels of AI usage, obtained through commit-level attribution, telemetry from assistant vendors, or alliances with organisations willing to disclose their tooling, would allow replacement of the temporal proxy with a measured treatment variable. Employing a multi-tool design would reduce dependence on a single ruleset and enable estimation of tool-specific false-positive rates in the production versus non-production code dimension identified here. Finally, research on organisational disposition should follow findings through organisations'

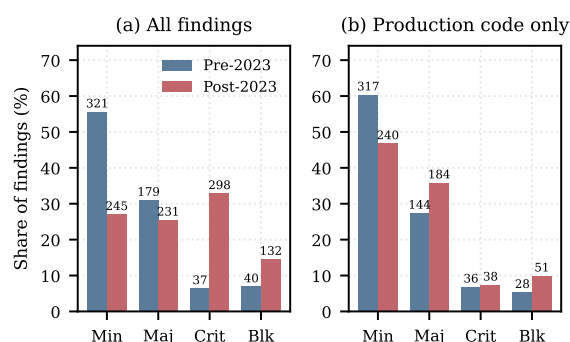


Figure 3. Severity distribution. (a) All findings. (b) Production code only. Bar labels are counts.

existing scanners and work-management systems over a shared observation window. Newly created research instances alone cannot establish whether, where, or how organisations review scanner output.

References

- Aloraini, B., Nagappan, M., German, D. M., Hayashi, S., & Higo, Y. (2019). An empirical study of security warnings from static application security testing tools. *Journal of Systems and Software*, 158, 110427. <https://doi.org/10.1016/j.jss.2019.110427>
- Avgeriou, P., Kruchten, P., Ozkaya, I., & Seaman, C. (2016). *Managing technical debt in software engineering (dagstuhl seminar 16162)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. <https://doi.org/10.4230/DAGREP.6.4.110>
- Charoenwet, W., Thongtanunam, P., Pham, V.-T., & Treude, C. (2024). An empirical study of static analysis tools for secure code review. *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 691–703. <https://doi.org/10.1145/3650212.3680313>
- Chidukwani, A., Zander, S., & Koutsakis, P. (2022). A survey on the cyber security of small-to-medium businesses: Challenges, research focus and recommendations. *IEEE Access*, 10, 85701–85719. <https://doi.org/10.1109/ACCESS.2022.3197899>
- Croft, R., Xie, Y., Zahedi, M., Babar, M. A., & Treude, C. (2022). An empirical study of developers' discussions about security challenges of different programming languages. *Empirical Software Engineering*, 27(1), 27. <https://doi.org/10.1007/s10664-021-10054-w>
- Cunningham, W. (1993). The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2), 29–30. <https://doi.org/10.1145/157710.157715>
- DeBellis, D., Storer, K., Lewis, A., Good, B., Villalba, D., Maxwell, E., & Castillo, K. (2024). *Dora accelerate state of devops 2024 report* (tech. rep.). DORA, Google. Retrieved August 18, 2026, from <https://dora.dev/research/2024/dora-report/2024-dora-accelerate-state-of-devops-report.pdf>
- Devos, J., Van Landeghem, H., & Deschoolmeester, D. (2012). Rethinking IT governance for SMEs. *Industrial Management & Data Systems*, 112(2), 206–223. <https://doi.org/10.1108/02635571211204263>
- European Commission. (2003). Commission recommendation of 6 may 2003 concerning the definition of micro, small and medium-sized enterprises.
- Fu, Y., Liang, P., Tahir, A., Li, Z., Shahin, M., Yu, J., & Chen, J. (2025). Security weaknesses of copilot-generated code in GitHub projects: An empirical study. *ACM Transactions on Software Engineering and Methodology*, 34(8), 1–34. <https://doi.org/10.1145/3716848>
- GitHub. (2026). *Code scanning with CodeQL*. GitHub. Retrieved July 20, 2026, from <https://docs.github.com/code-security/code-scanning/introduction-to-code-scanning/about-code-scanning-with-codeql>
- Khoury, R., Avila, A. R., Brunelle, J., & Camara, B. M. (2023). How secure is code generated by ChatGPT? *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2445–2451. <https://doi.org/10.1109/SMC53992.2023.10394237>
- Letouzey, J.-L., & Ilkiewicz, M. (2012). Managing technical debt with the SQALE method. *IEEE Software*, 29(6), 44–51. <https://doi.org/10.1109/MS.2012.129>
- Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101, 193–220. <https://doi.org/10.1016/j.jss.2014.12.027>
- Martinez, J., Quintano, N., Ruiz, A., Santamaria, I., De Soria, I. M., & Arias, J. (2021). Security debt: Characteristics, product life-cycle integration and items. *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*, 1–5. <https://doi.org/10.1109/TechDebt52882.2021.00009>
- Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2022). Asleep at the keyboard? assessing the security of GitHub copilot's code contributions. *2022 IEEE Symposium on Security and Privacy (SP)*, 754–768. <https://doi.org/10.1109/SP46214.2022.9833571>
- Pedro, R., Castro, D., Carreira, P., & Santos, N. (2025, January 27). From prompt injections to SQL injection attacks: How protected is your LLM-integrated web application? <https://doi.org/10.48550/arXiv.2308.01990>
- Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023). Do users write more insecure code with AI assistants? *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2785–2799. <https://doi.org/10.1145/3576915.3623157>
- Recupito, G., Pecorelli, F., Catolino, G., Lenarduzzi, V., Taibi, D., Di Nucci, D., & Palomba, F. (2024). Technical debt in AI-enabled systems: On the prevalence, severity, impact, and management strategies for code and architecture. *Journal of Systems and Software*, 216, 112151. <https://doi.org/10.1016/j.jss.2024.112151>
- Rindell, K., Bernsmed, K., & Jaatun, M. G. (2019). Managing security in software: Or: How i learned to stop worrying and manage the security technical debt. *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 1–8. <https://doi.org/10.1145/3339252.3340338>
- Sandoval, G., Pearce, H., Nys, T., Karri, R., Garg, S., & Dolan-Gavitt, B. (2023). Lost at c: A user study on the security implications of large language model

- code assistants. *Proceedings of the 32nd USENIX Conference on Security Symposium*.
- Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 28). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2015/file/86df7dcfd896fcdf2674f757a2463eba-Paper.pdf
- SonarSource. (2026a). *Managing security hotspots*. SonarSource. Retrieved July 20, 2026, from <https://docs.sonarsource.com/sonarqube-server/user-guide/security-hotspots>
- SonarSource. (2026b). Metric definitions: Understanding measures and metrics [Accessed 2026].
- Veracode. (2025). *Genai code security report* (tech. rep.).
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Springer Berlin Heidelberg. <https://doi.org/10.1007/978-3-642-29044-2>