The Indonesian Journal of Computer Science
www.ijcs.net
Volume 15, Issue 5, October 2026
https://doi.org/10.33022/ijcs.v15i5.5244

Visibility Over Label: Technical Debt Tracking, Developer Experience, and Software Outcomes in South African Software Engineering

Tshipuke Vhahangwele[1], Bassey Isong[2], Elegbeleye Femi[3]
tshipukevhahangwele@gmail.com[1], Bassey.Isong@nwu.ac.za[2], phernet123@gmail.com[3]
[1,2,3]North West University, South Africa

Article Information

Received : 11 Sep 2026
Revised : 12 Oct 2026
Accepted : 25 Oct 2026

Keywords

technical debt management; developer experience; technical debt tracking; software maintainability; mixed-methods research

Abstract

Empirical research on technical debt (TD) management has largely examined repository data or small qualitative samples. Less is known about how proactive, reactive, and mixed management approaches relate to software development outcomes within the same group of practitioners. Developer experience has also received less attention than technical outcomes such as maintainability. In this study, we surveyed 31 software practitioners in South Africa: 13 used proactive management, 9 reactive management, and 9 a mixed approach. The questionnaire measured five outcomes such as morale, maintainability, confidence, frustration, and motivation, and included open-ended questions about TD management. Kruskal-Wallis tests showed significant differences among the three approaches for all five outcomes. Post-hoc Mann-Whitney tests found differences between reactive teams and both proactive and mixed teams. No significant difference was found between proactive and mixed teams. TD tracking was also associated with better outcomes, including confidence and motivation. The qualitative responses described slower development, software quality problems, and effects on morale. Practitioners also reported refactoring, structured TD management, and automation as ways of addressing debt. Overall, the findings indicate that tracking and managing TD may be more important than the management approach label itself. The results also point to developer experience as an important outcome when evaluating TD management.

Attribution-ShareAlike 4.0 International License

A. Introduction

Technical debt (TD) can occur in the software development environment when teams adopt insufficient solutions due to short-term delivery targets, resulting in more rework later. The metaphor characterises this trade-off as an immediate reduction in effort followed by extra corrective labour, known as "interest" [1], [2]. Poorly managed TD can affect both software and developers, contributing to slower delivery, more defects, lower dependability, higher maintenance costs, irritation, low morale, and burnout [3], [4]. TD can also be intentionally incurred and handled strategically. The tracking, communication, and repayment of the debt then determine its consequences [1], [2], [5], [6] , and [7].

Despite a growing body of literature on the technical and organisational dimensions of technical debt, three gaps motivate the present study. First, much of the empirical evidence on TD management (TDM) is either repository-scale, relying on automated detection of self-admitted technical debt in code comments and issue trackers [8], [9], [10], or small-scale and qualitative, relying on interviews with a handful of teams or organisations [11], [12]; even where managers report that TDM is essential, most organisations still lack a formal process for it [12], and few studies quantitatively link a specific management approach, whether proactive, reactive, or mixed, to a broad set of software development outcomes measured within the same sample. Second, while automated identification, measurement, and prediction of technical debt have advanced, prevention and communication together account for under 8% of intelligent TDM research [13], leaving the activities that arguably determine whether debt is actually managed, rather than merely detected, comparatively under-researched; this imbalance persists even in generative-AI-assisted TDM, where continuous monitoring was absent and human oversight was operationalised in only six of 29 reviewed studies [14]. Third, the developer experience dimension of technical debt, including morale, confidence, motivation, and frustration, is often treated as a secondary outcome rather than a primary variable of interest, even though practitioner evidence suggests it is one of the most consistently reported consequences of unmanaged debt [5], [6], [7].

The present study addresses these gaps by examining how technical debt management approaches relate to software development outcomes among practitioners working across a range of roles, team sizes, and industries in South Africa. A mixed-methods design was adopted, combining a structured online questionnaire that captured quantitative measures of software development outcomes (including morale, confidence, maintainability, motivation, frustration, bug frequency, development slowdown, cycle time, and issue frequency) with open-ended questions that captured practitioners' lived experience of managing technical debt. Specifically, the study sought to examine whether software development outcomes differ significantly among teams adopting proactive, reactive, and mixed technical debt management approaches; to examine the relationships between technical debt management practices, particularly technical debt tracking, and software development outcomes; and to explore, through practitioners' own accounts, the experiences, challenges, and strategies associated with managing technical debt in practice.

This study contributes to the literature in three ways, corresponding to the gaps identified above. First, it provides one of the first quantitative comparisons of proactive, reactive, and mixed technical debt management approaches against a broad set of software development outcomes measured within a single sample, rather than relying on repository-scale proxies or small qualitative accounts alone. Second, by treating technical debt tracking as a variable of interest in its own right, it offers empirical evidence on a specific TDM practice rather than TDM as an undifferentiated whole. Third, by measuring morale, confidence, motivation, and frustration alongside conventional technical outcomes such as maintainability and cycle time, it treats developer experience as a primary rather than a secondary outcome of technical debt management. Combining quantitative and qualitative evidence from the same sample lets this study capture something neither method alone would show: how TD management approaches shape technical outcomes and human ones together, not in isolation.

Section B covers the relevant TD literature. Section C sets out the methodology. Section D reports the quantitative and qualitative results. Section E discusses those findings against existing work, considers what they mean practically and theoretically, and addresses the study's limitations. Section F closes the paper and points to specific directions for future work.

B. Related Works

This section discusses some of the connected studies. In software development, TD is viewed as a future cost of adopting a technically poor solution for immediate advantage, with saved effort portrayed as principal and subsequent repair as interest [1], [2]. Previous research has established that technology development is intrinsically socio-technical. It is difficult to identify and repay once distributed across components [1]; developers discount its future cost [2], and how it plays out is determined by sprint capacity [11], TD's social, process, documentation, and requirements dimensions [5], and whether accountability structures are democratic or bureaucratic [6]. The human cost is direct: lower morale due to time spent on debt-related work [3]. This expense adds to a chronic visibility gap; managers consider TDM as necessary but seldom formalise it [12], acceptable debt shifts during a startup's growth. These works position TD as a decision problem shaped by time preference, delivery capacity, growth stage, accountability, and visibility, not a purely technical one.

A second set of studies focuses on automated TD detection. These studies primarily identify self-admitted technical debt (SATD) in comments, issues, and commits using classification algorithms, attention-based highlighting, and large language model (LLM) data [9], [10], [15]. While evaluations of detection techniques reveal little overlap and inadequate methodological information for evaluating external validity [16], [17], other research has looked at explainability and TD prediction [15], [18]. In contrast to 5.4% for preventive and 1.8% for communication, intelligent TDM supported detection in 71.2% of instances across 111 primary investigations [19]. Even among seasoned companies, TDM maturity varies; restructuring was the most popular repayment strategy (37.2%), while short-term delivery pressure was the primary cause of unpaid debt (26.4%) [7], [20]. Additionally, there was a stronger correlation between frustration and debt

related to requirements and design [7]. There is still little or no evidence to support integrated frameworks and ML-assisted procedures [21], [22]. Identification is prioritised over prevention and monitoring in generative AI research, and human oversight is rarely operationalized [14], [23]. According to a 2026 study, 24.2% of AI-introduced issues in over 6,000 repositories remained unsolved as of the most recent revision [24]. Clear tracking and shared accountability are associated with increased developer confidence, whereas opaque, blame-oriented procedures are associated with higher cognitive and coordination demands [5], [6], and [7].

Similarly, data, configuration, model selection, training, and deployment can all contribute to ML system debt. Studies attempting to discover and describe this debt have had little success, with issues about F1-scores, sample validity, and repayment attribution [25], [26], [27], [28]. Security research complicates matters further: SATD can provide a valuable signal but may also introduce weaknesses, and documented detection-to-prediction pipelines are rare [29], [30]. The primary gap in ML, generative AI, and security research is not discovering novel kinds of debt, but rather verifying, communicating, and retiring debt throughout its lifecycle [14], [30]. Table 1 highlights the investigations based on emphasis, technique, and findings.

Table 1. Summary of Related Works

Study	Focus	Methodology	Sample	Finding	Limitation
[1]	Architectural TD	Grounded theory and focus groups	18 interviews; two focus groups with nine practitioners	Integrated symptoms, causes, consequences, management, and communication.	Qualitative and context-dependent evidence.
[2]	Temporal discounting	Online choice questionnaire	33 developers in two companies	Future TD costs were widely discounted, with strong individual variation.	Small sample and hypothetical choices.
[12]	Managerial TDM	Interviews and focus group	16 interviews; three-person focus group	Managers valued TDM, but almost no organisation had a formal process.	Small, management-centred sample.
[8]	Open-source TDM	Repository mining and manual coding	35,278 issues in 3,598 repositories	TD issues resolved more slowly; management was often inconsistent or abandoned.	Captures only explicitly labelled debt.
[9]	Explainable SATD detection	Attention-based neural benchmark	445,365 comments from 20 projects	HATD classified SATD and highlighted influential words.	Attention is not a causal explanation; comments only.
[10]	Cross-artefact SATD	Multi-task BERT evaluation	>9,000 positive comments plus issue,	MT-BERT-SATD averaged F1 = 0.712; artefact transfer was	Curated open-source datasets limit transfer claims.

			PR, and commit data	harder than language transfer.	
[18]	High-debt Java classes	Multi-tool convergence and ML	18,857 classes from 25 Java projects; Commons IO case	F2 was about 0.79 while inspecting roughly 10% of classes.	Tool-defined ground truth and Java-only case.
[31]	TD forecasting	Longitudinal predictive modelling	1,850 weekly snapshots from 15 repositories	MAPE was about 1.4% at one week and 4% at five weeks.	No naive baseline; possible target circularity.
[16]	TD identification tools	Survey and empirical comparison	Catalogue and Java-project evaluation	eXcomment and DebtHunter had little overlap and low precision or recall.	Small oracle, low agreement, and class imbalance.
[19]	Intelligent TDM techniques	Systematic mapping study	111 primary studies	71.2% addressed identification; prevention and communication received 5.4% and 1.8%.	Evidence limited to publications from 2010–2020.
[7]	TD payment	Cross-country practitioner survey	653 practitioners in six countries	Refactoring was most common; short-term goals were the leading reason for non-payment.	Self-report based on one salient debt item.
[21]	Scientific-software TDM	Framework development and pilot evaluation	Five scientific applications	Most recommendations were applied and reportedly eased later maintenance.	Domain-specific and evaluated by project participants.
[14]	Generative AI for TDM	Systematic literature review	29 studies	Prompting and identification dominated; only six studies operationalised human oversight.	Screening and quality-assessment limitations.
[32]	SATD in ML software	Comparative repository mining	318 ML and 318 non-ML repositories	ML projects showed more raw SATD and additional configuration and test categories.	Unmatched repositories and detector-validity concerns.
[30]	Security-related SATD	Mixed-methods mining and survey	8,812 SATD items; 222 contributors	201 security instances mapped to 25 CWE categories, including eight MITRE Top 25 weaknesses.	Open-source and self-reported evidence.

Table 1 illustrates that whereas management studies look at decisions, accountability, payment, and organisational settings [2], [6], [7], [8] , and [21], text- and metric-based research mostly evaluate detection or prediction performance[9], [17]. While practitioner studies offer context but frequently employ small, cross-sectional, or self-reported samples, repository studies offer scale but provide little insight into how TDM tactics impact practitioners. There is still little longitudinal data connecting particular TDM techniques to developer experience and maintainability. Additionally, research prioritises debt detection over human oversight, communication, monitoring, and prevention [14], [19]. The current study, which focuses on debt tracking and practitioner experiences, is motivated by these gaps and looks at proactive, reactive, and mixed TDM tactics.

C. Methodology

In order to investigate the connection between TD management strategies and software development results, this study employed a mixed-methods strategy. Both quantitative and qualitative data were gathered using a structured online survey. During interpretation, the two types of data were combined after being analysed independently. Contextual explanations of how practitioners handle technical debt in software projects were offered by the qualitative responses. The quantitative analysis looked at correlations between TD-related parameters and compared development results among management strategies. Software experts from South African software development companies engaged in system design, development, testing, deployment, or maintenance made up the target population. Software developers, technical leads, engineering managers, and product managers were among the several software engineering roles represented by the participants. They had real-world experience with technical debt and worked for several software companies. In order to find participants with pertinent experience, purposeful sampling was employed. Professional software engineering forums and online platforms were used for recruitment. For the analysis, 31 valid replies in all were kept.

Prior to the start of data collection, the Northwest University Research Ethics Committee granted ethical permission. Over the course of six weeks, Google Forms was used to administer the survey. There were both closed-ended and open-ended questions in each of its four sections. Participants' duties, team size, software development methodologies, industrial sector, and codebase features were all covered in the first segment. In the second, TD management techniques were categorised as either proactive, reactive, or mixed. The third dealt with the results of software development. Morale, confidence, maintainability, motivation, frustration, bug frequency, delay in development, cycle length, and issue frequency were among them. Participants were asked about their technical debt management experiences, difficulties, and recommendations in the last section. Participation was voluntary. No personally identifiable information was gathered, and responses were anonymous and private.

IBM SPSS Statistics v30 was used to analyse quantitative data. To characterise the software development techniques and participant profile, frequencies and percentages were computed. Bar charts and pie charts were also used to present these descriptive results. Because the dependent variables were measured on

ordinal scales and normality assumptions could not be guaranteed given the limited sample size, non-parametric statistical approaches were used. To find out if software development outcomes varied significantly amongst teams using proactive, reactive, and mixed technical debt management approaches, the Kruskal-Wallis H test was employed.

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{k} n_i R_i^2 - 3(N+1) \quad \text{.....(1)}$$

Where H = Kruskal-Wallis statistic, N = total sample size, k = number of groups, R_i = sum of ranks for group I, n_i = sample size for group i. Whenever statistically significant differences were identified ($p < 0.05$), pairwise comparisons were conducted using the Mann-Whitney U test with Bonferroni correction to identify which groups differed significantly.

We also used the Mann-Whitney U test to compare two independent technical debt management groups at a time following significant Kruskal-Wallis results.

$$U = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1 \quad \text{.....(2)}$$

Where U = Mann-Whitney statistic, n_1 = size of Group 1, n_2 = size of Group 2, R_1 = sum of ranks for Group 1

To control for Type I error associated with multiple pairwise comparisons, the Bonferroni-adjusted significance level was calculated as

$$\alpha_{adj} = \frac{\alpha}{m} \quad \text{.....(3)}$$

where m represents the number of pairwise comparisons. With three comparisons, the adjusted significance threshold was:

$$\alpha_{adj} = \frac{0.05}{3} = 0.0167 \quad \text{.....(4)}$$

Effect Size

For statistically significant Kruskal-Wallis tests, practical significance was assessed using Eta Squared (η^2):

$$\eta_H^2 = \frac{H-k+1}{n-k} \quad \text{.....(5)}$$

Where n = total sample size, H = Kruskal-Wallis statistic, and k = number of comparison groups. Cohen's criteria were used to interpret the effect sizes, which were roughly 0.01, 0.06, and 0.14, indicating minor, medium, and large effects, respectively.

In the same vein, Spearman's rank-order correlation coefficient was used to analyse relationships between technical debt management variables and software development outcomes because the variables were ordinal and did not require assumptions of normality.

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \quad \text{.....(6)}$$

where n is the number of paired data, d_i is the difference between paired ranks, and ρ is the Spearman's correlation coefficient. Correlation coefficients were interpreted based on their size and direction, and statistical significance was assessed at the 0.05 and 0.01 significance levels.

The six-phase theme analysis method developed by Braun and Clarke was used to analyse the answers to the open-ended questions. To become comfortable with the facts, the answers were initially read several times. Initial codes were generated and grouped into broader categories representing common ideas. Related codes were then combined into overarching themes, which were reviewed, refined, and clearly defined. Theme frequencies were calculated to identify the most commonly reported experiences regarding technical debt management. Representative participant quotations were used to support each identified theme. Finally, the qualitative findings were compared with the quantitative results to identify areas of convergence and divergence, thereby providing a richer understanding of the study findings. Additionally,

D. Results

This section presents the findings of the study. It begins with the demographic and contextual profile of the sample, followed by the quantitative results of the Kruskal-Wallis and post-hoc Mann-Whitney comparisons across proactive, reactive, and mixed technical debt management approaches, and Spearman's rank-order correlations between technical debt tracking and software development outcomes. The section concludes with the thematic analysis of the open-ended survey responses.

D.1 Demographics

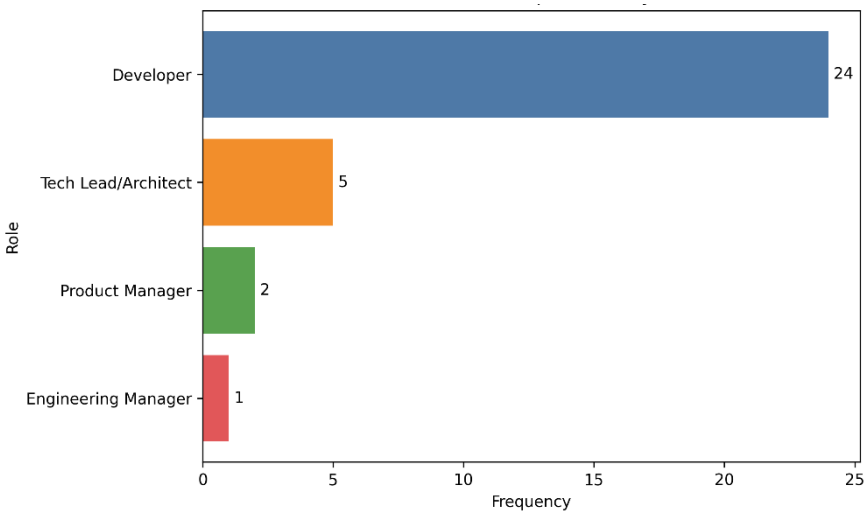

Figure 1. Distribution of respondents by role

Illustrated in the bar chart, the majority of respondents in this study were software developers, accounting for approximately 75% of the sample. This was followed by Tech Leads/Architects, while Product Managers and Engineering Managers constituted a relatively small proportion of the respondents.

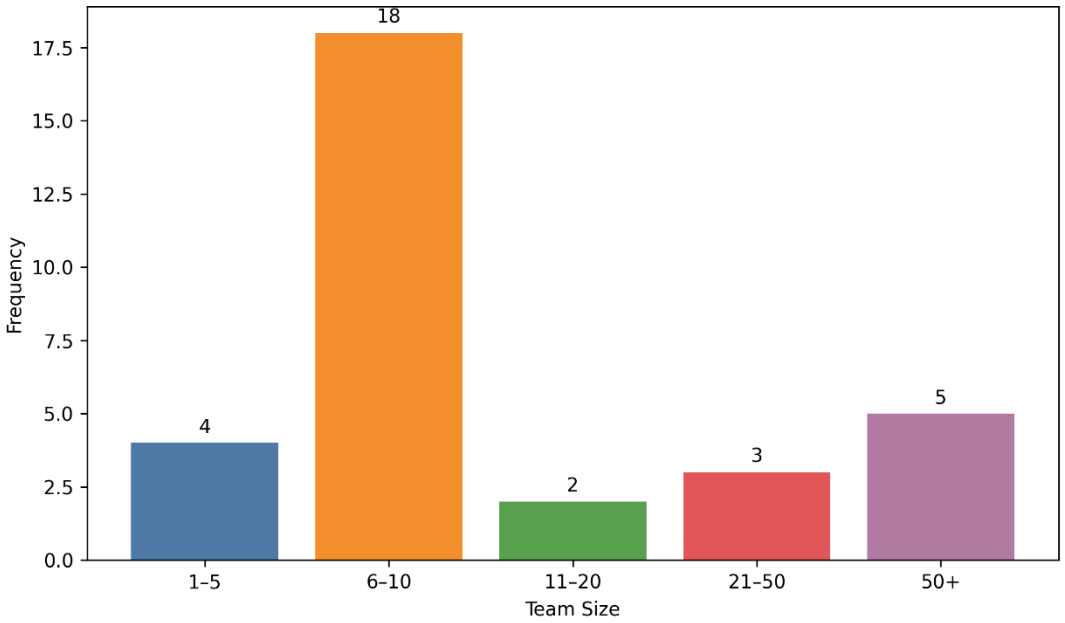

Figure 2. Distribution of respondents by team size

As illustrated in the bar chart, most respondents reported working in teams consisting of 6–10 members, making this the most common team size within the sample. Smaller teams comprising 1–5 members were less represented, while relatively few respondents indicated working in teams of 11–20 or 21–50 members. A notable proportion of respondents also reported working in teams with more than 50 members.

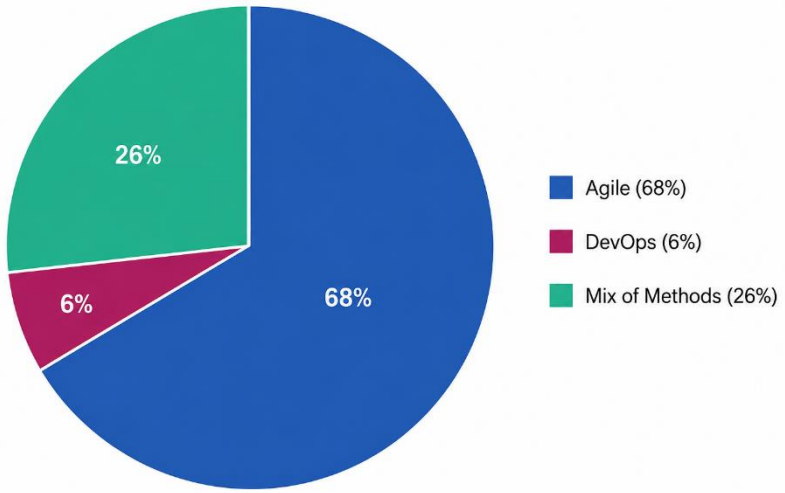

Figure 3. Software Development Process used by Respondents

As illustrated in the pie chart, the majority of respondents reported using Agile as their primary software development process, accounting for 68% of the sample. This was followed by a mix of methods, which represented 26% of respondents, while only 6% indicated using DevOps as their primary development approach.

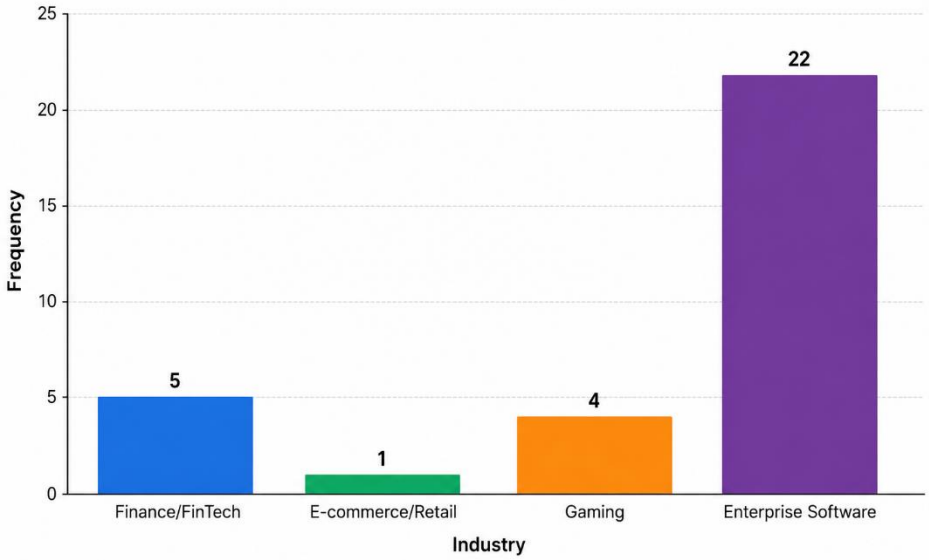

Figure 4. Distribution of respondents by Industry

As illustrated in the bar chart, the majority of respondents were drawn from the Enterprise Software industry, which accounted for the largest proportion of the sample. This was followed by respondents from the Finance/FinTech and Gaming sectors, while the E-commerce/Retail industry was represented by only a small number of participants.

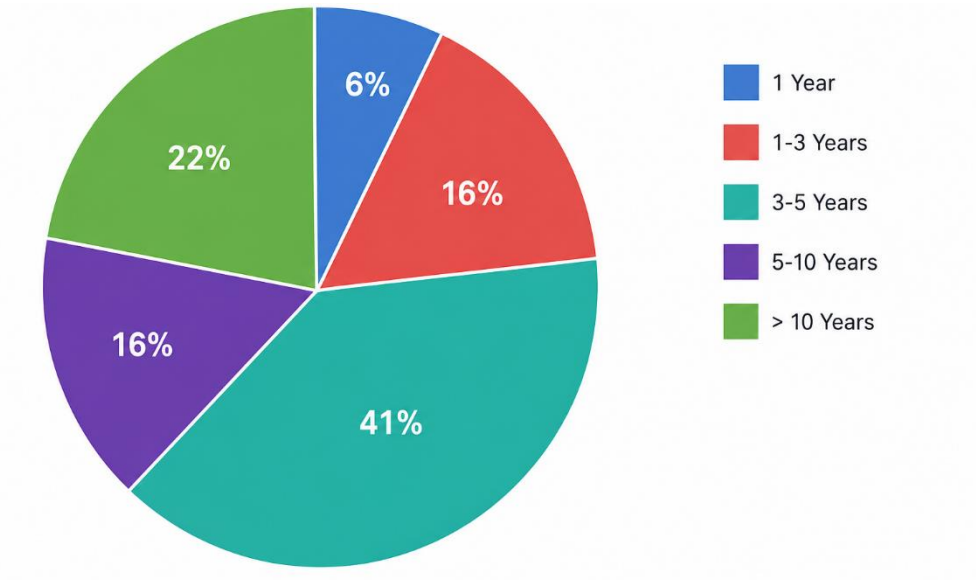

Figure 5. Codebase Age

As illustrated in the pie chart, the majority of respondents reported working with codebases that were between 3 and 5 years old, accounting for 41% of the sample. This was followed by codebases older than 10 years, which represented 22% of respondents. Codebases aged 1-3 years and 5-10 years each accounted for 16% of the sample, while only 6% of respondents reported working with codebases that were less than one year old.

D.2 Kruskal-Wallis Tests

Table 2. Kruskal-Wallis H test results for software development outcomes across technical debt management approaches

Outcome	Proactive Mean Rank (n=13)	Reactive Mean Rank (n=9)	Mixed Mean Rank (n=9)	H(2)	p
Morale	11.50	23.89	14.61	12.944	.002
Confidence	21.73	6.83	16.89	16.671	<.001
Maintainability	10.35	21.06	19.11	9.647	.008
Motivation	20.92	7.33	17.56	14.391	<.001
Frustration	11.46	24.22	14.33	12.195	.002

Kruskal-Wallis H tests revealed statistically significant differences across proactive, reactive, and mixed technical debt management approaches for all five outcome variables (Table 2). In every case, teams following a reactive approach occupied the least favourable mean rank: highest for morale and frustration, and lowest for confidence and motivation, with maintainability following the same reactive-disadvantage pattern. This suggests that the manner in which technical debt is managed, rather than the presence of debt itself, has a consistent influence on both the technical and human dimensions of software development. Pairwise comparisons identifying which specific approaches differed are reported in Table 3.

D.3 Post-Hoc Mann-Whitney Comparisons

Table 3. Post-hoc Mann-Whitney U pairwise comparisons (Bonferroni-adjusted α = .0167)

Outcome	Comparison	U	p	Decision
Morale	Proactive vs Reactive	14.000	.001	Significant
Morale	Proactive vs Mixed	44.500	.262	Not significant
Morale	Reactive vs Mixed	14.000	.007	Significant
Confidence	Proactive vs Reactive	4.500	<.001	Significant
Confidence	Proactive vs Mixed	38.000	.105	Not significant
Confidence	Reactive vs Mixed	12.000	.007	Significant
Maintainability	Proactive vs Reactive	19.000	.006	Significant
Maintainability	Proactive vs Mixed	34.500	.571	Not significant
Maintainability	Reactive vs Mixed	24.500	.016	Significant
Motivation	Proactive vs Reactive	10.500	<.001	Significant
Motivation	Proactive vs Mixed	42.500	.209	Not significant
Motivation	Reactive vs Mixed	10.500	.004	Significant
Frustration	Proactive vs Reactive	11.000	<.001	Significant
Frustration	Proactive vs Mixed	47.000	.401	Not significant

Frustration	Reactive vs Mixed	14.000	.012	Significant

The pattern of pairwise differences was consistent across all five outcome variables (Table 3): teams following a reactive approach differed significantly from teams following both a proactive and a mixed approach on every outcome, while proactive and mixed approaches were never statistically distinguishable from one another. This indicates that the critical distinction for software development outcomes is not the specific balance of proactive and reactive practice a team adopts, but whether technical debt is addressed only after it becomes a visible problem.

D.4 Spearman's Rank-Order Correlations

Table 4. Spearman's rank-order correlations between technical debt tracking and software development outcomes

Outcome	ρ	p
Morale	-.550	.001
Confidence	.810	<.001
Maintainability	-.360	.047
Motivation	.783	<.001
Frustration	-.555	.001
Cycle time	-.394	.028

Spearman's rank-order correlations show that clearer technical debt tracking was associated with better outcomes across the board (Table 4): tracking was strongly positively associated with developer confidence and motivation, and significantly associated with lower frustration, reduced morale-related strain, and improved maintainability and cycle time. Unlike the broad management-approach comparisons in Table 2 and Table 3, these correlations point to a specific practice, tracking, as the aspect of technical debt management most consistently related to better outcomes for both the system and the people maintaining it.

D.5 Open-Ended Responses

Table 5. Themes, frequencies, and supporting evidence from the thematic analysis

Theme	Frequency (n)	Finding	Representative evidence
Technical debt reduces development speed	18	Increased development effort, delayed feature implementation, and slower delivery were associated with legacy code, outdated dependencies, poorly structured modules, and interdependent components.	R1: “A three-day feature took six days because of the complexity introduced by the existing structure.”
Technical debt affects software quality	15	Technical debt was associated with defects, system failures, performance problems, and unstable releases.	R2: “A bug was found lingering through multiple versions.”
Technical debt impacts team	12	Participants reported frustration, stress, overtime,	R3: “The entire team was demoralised.”

morale and well-being		reduced confidence, and declining morale when dealing with technical debt.	
Refactoring as a mitigation strategy	14	Teams used dedicated refactoring activities and refactoring spikes to address accumulated technical debt.	R1: "We dedicated time each sprint to refactor problematic areas."
Need for structured technical debt management	17	Participants recommended formal mechanisms for identifying, documenting, tracking, prioritising, and managing technical debt.	R2: "Integrate technical debt into sprint planning."
Automation and tool support	10	Automation, static analysis, code-quality tools, automated refactoring, and CI/CD practices were identified as mechanisms for monitoring and reducing technical debt.	R2: "Use SonarQube to consistently track code quality metrics."

The thematic analysis produced six themes concerning the effects and management of technical debt. Reduced development speed was the most frequent (n = 18), with respondents describing delays in feature delivery, longer implementation times, and additional maintenance and debugging work. Software quality followed (n = 15), with accumulated debt associated with recurring defects, system failures, performance problems, and unstable releases. Human effects were also obvious (n = 12), particularly in complaints of dissatisfaction, stress, overtime, and low morale. Refactoring took place in 14 coding instances and was often described as a method of addressing issue regions. Debt backlogs, sprint planning, and issue tracking were among 17 instances of structured management approaches. Automation and continuous integration were cited less frequently (n = 10), mostly in terms of code quality monitoring, static analysis, automated refactoring, and CI/CD techniques. Overall, the data indicate that technical debt has an impact on more than just the code; it also influences development work, software quality, and developer experience.

E. Discussion

The results are discussed in this section. Using practitioner perspectives to contextualise the quantitative results, we investigated the relationship between TD management and development outcomes among South African SE professions. The results show that management practices are associated with morale, maintainability, confidence, frustration, and motivation, with reactive approaches linked to poorer outcomes.

E.1 Management approach and software development outcomes

We discovered substantial variations in morale, maintainability, confidence, frustration, and motivation for proactive, reactive, and mixed approaches using the Kruskal-Wallis test. Proactive and mixed teams were statistically identical, whereas reactive teams varied significantly from proactive or mixed teams across all outcome variables, according to post-hoc Mann-Whitney tests. This pattern indicates that whether debt is addressed until after it becomes an obvious issue is

more important for software development results than the precise combination of proactive and reactive practices a team chooses. This is consistent with the findings of Wiese and Borowa, who found that IT managers understood the value of TDM but lacked formal procedures [12], and Freire et al.'s survey, which indicates that short-term delivery objectives and a lack of organisational engagement are the primary causes of unpaid debt [7]. In both situations, the mechanism by which debt results in worse outcomes for practitioners is not the simple existence of debt but rather the lack of a proactive, organised approach.

This perspective is supported by the Spearman correlations at the individual practice level rather than the overall strategy. TD tracking was strongly and positively associated with developer confidence and motivation, and significantly associated with lower frustration, alongside improved cycle time and maintainability. This result is in line with the socio-technical perspective of technical debt presented by Graetsch et al. [11] and Verdecchia et al. [1]. It implies that making debt apparent could be more important for managing it than just having it. The results also support the accountability research of Ghanbari et al. [6]. When debt is tracked and recorded, team members may share accountability for resolving it instead of just assigning blame in times of need. Because visibility makes accountability and when action is necessary clear, it may help explain the relationship between debt tracking, frustration, and confidence.

E.2 Comparison with existing empirical studies

These results align with previous studies on technical debt. Besker et al. [3] also discovered a connection between TD and worse developer morale using time-diary data and interviews from 43 developers. The current survey expands on that finding by demonstrating a statistically significant difference in morale specifically tied to management approach ($H(2) = 12.944$, $p = .002$), with reactive teams reporting the lowest morale. The refactoring-as-primary-strategy theme identified here also echoes Freire et al.'s [7] cross-country survey of 653 practitioners, in which refactoring was the most frequently reported payment practice (37.2%); the convergence across a sample roughly twenty times larger suggests the finding is not an artefact of this study's small size. The absence of formal TDM processes reported by the IT managers in Wiese and Borowa's interviews [12] is also consistent with the pattern observed here, where reactive teams, those most likely to lack a structured process, showed the weakest outcomes across all five variables measured.

Other comparisons point to differences in scope rather than contradiction. Li et al. [8] used repository-mined GitHub issues to show that technical-debt-labelled issues remain open longer under inconsistent management; that behavioural, artefact-level evidence and the self-reported, perception-level evidence collected here rely on different data sources but arrive at the same conclusion, that inconsistent management is associated with worse outcomes, which strengthens confidence in the result through methodological triangulation. Kassab et al.'s [33] large-scale mining of issue-tracker comments found that developer frustration intensifies around specific debt types, such as design and requirements debt, rather than around management approach as such; this study did not code technical debt by type, so it cannot say whether the frustration difference observed

between reactive and proactive teams (H(2) = 12.195, p = .002) is driven by the same debt categories identify, or by the process itself. Similarly, Aldaeej and Seaman [34] found that acceptable technical debt decisions shift across a startup's inception, stabilisation, and growth stages; because this study did not measure organisational age or stage, it cannot determine whether the reactive-approach penalty observed here would hold, strengthen, or weaken at different points in a company's lifecycle. Table 6 summarises these comparisons.

Table 6. Comparison of the present study's findings with prior empirical studies

Prior study	Prior finding	Present study's finding	Relationship
Besker et al. [3]	TD linked to reduced developer morale (interviews and time-diary data, n=43)	Morale differs significantly by approach (H(2)=12.944, p=.002); reactive lowest	Convergent, extends
Freire et al. [7]	Refactoring most common payment practice (37.2%, n=653)	Refactoring identified as the primary mitigation theme	Convergent
Wiese and Borowa [12]	Managers value TDM but rarely formalise it	Reactive (less structured) teams show weakest outcomes on all five variables.	Convergent
Li et al. [8]	TD-labelled issues stay open longer under inconsistent management (repository mining)	Inconsistent management associated with worse outcomes (self-report)	Convergent via triangulation
Kassab et al. [33]	Frustration intensifies around specific debt types, design, requirements (issue-tracker mining)	Frustration differs significantly by approach (H(2)=12.195, p=.002)	Complementary; open question
Aldaeej and Seaman [34]	Acceptable TD decisions shift across startup stages	Organisational stage not measured	Scope difference; unresolved

E.3 Convergence between the quantitative and qualitative findings

The thematic analysis of the open-ended responses provides a practitioner-level explanation for the statistical patterns above. Theme 1 (technical debt reduces development speed) and Theme 2 (technical debt negatively affects software quality) describe the technical consequences of debt that prior repository-scale studies have documented at scale, for example Li et al.'s finding that technical-debt-labelled issues remain open longer than other issues and are frequently managed inconsistently [8]. Theme 3 (impacts on team morale and developer well-being) gives qualitative texture to the quantitative morale, confidence, and frustration results, and is consistent with the human-cost emphasis found in the accountability and temporal-discounting literature [2], [6]. Themes 4 to 6, refactoring as the primary mitigation strategy, the need for structured technical debt management, and automation and continuous-integration support, map closely onto the practices whose presence or absence differentiated proactive and mixed teams from reactive teams in the quantitative results. Respondents' own recommendations, a dedicated debt backlog, integration into sprint planning, and static-analysis tooling, are precisely the kind of visibility- and tracking-oriented practices that the Spearman correlations associate with better outcomes, and they echo Freire et al.'s finding that refactoring is the most

frequently reported payment practice among practitioners [7] and Pandya's field evidence that combining debt-tagging, monitoring, and AI-assisted refactoring can measurably reduce debt density and improve maintainability [22].

E.4 Implications for technical debt management research and practice.

For practitioners, the results offer a relatively direct message: the specific label a team gives its approach matters less than whether technical debt is made visible and addressed before it accumulates into a crisis. Investing in lightweight tracking, whether a backlog, a tagging system, or integration with existing issue trackers, appears to be associated with meaningfully better developer experience as well as better perceived maintainability, which supports calls in the literature for TDM to be treated as a socio-technical rather than a purely technical concern [1], [5], [11]. For researchers, the findings help address a gap identified by Albuquerque et al.'s systematic mapping, which found that identification dominates intelligent TDM research while prevention and communication receive comparatively little attention [19]. By directly relating a management approach to a broad set of developer-experience outcomes within a single sample, this study provides empirical support for prioritising prevention- and communication-oriented TDM research and tooling, rather than continuing to concentrate almost exclusively on detection and measurement, as reflected in the imbalance documented across the automated identification literature [9], [17]. The findings are also consistent with, and extend, Gosu et al.'s observation that generative-AI applications for TDM remain concentrated on identification, with prevention and continuous monitoring largely absent [14]: automation was welcomed by respondents in this study, but only in the context of a structured process into which it could be embedded, not as a substitute for one.

The mechanism is summarised in Figure 6. As indicated, management approach shapes whether debt is made visible and tracked, and it is this visibility, more than the formal label a team gives its approach, that in turn shapes both technical and developer-experience outcomes.

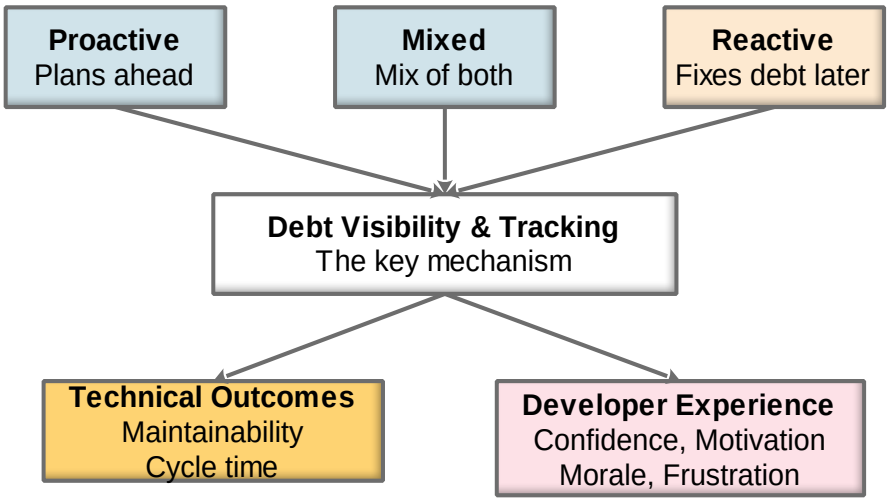

Figure 6. A visibility-mediated framework of TDM outcomes

F. Limitations

Several limitations qualify these conclusions. The sample of 31 valid responses, while sufficient for non-parametric group comparisons, is small, and the resulting subgroup sizes (9 to 13 per management-approach category) limit statistical power and the precision of the effect-size estimates; the findings should therefore be treated as indicative rather than definitive. The sample was recruited through purposive, self-selected participation in professional SE communities, which may under-represent practitioners who do not engage with such communities and may over-represent those with a particular interest in technical debt as a topic. All measures, including the classification of a team's management approach as proactive, reactive, or mixed, were self-reported by a single respondent per team rather than corroborated with repository data, team-level records, or other members of the same team, which introduces the possibility of common-method bias and single-informant distortion. The design is cross-sectional, so while significant associations were observed between management approach, tracking, and outcomes, causal direction cannot be established: it is equally plausible, for example, that teams with already higher morale and confidence find it easier to sustain proactive practices, as that proactive practices cause improved morale and confidence. This mirrors a gap noted in the wider literature, where longitudinal evidence connecting a specific management approach to both maintainability and developer experience remains scarce [8], [15], [21].

G. Conclusions

This study examined how proactive, reactive, and mixed approaches to technical debt management relate to software development outcomes among 31 practising software professionals in South Africa. Quantitative comparisons were combined with practitioners' accounts of how technical debt is managed in practice. The results show poorer outcomes under reactive management across morale, confidence, maintainability, motivation, and frustration. Technical debt tracking was also strongly associated with better developer experience, regardless of the broader management approach used by a team. The qualitative findings support these quantitative results and show that technical debt management extends beyond technical maintenance. How debt is identified, recorded, and addressed appears to matter for both the software and the people working with it. Future research can extend these findings in three areas. Longitudinal or repeated-measures studies could examine whether the relationships observed here remain stable over time and whether changes in technical debt management are followed by changes in developer experience and software outcomes. Repository-based measures could also be combined with practitioners' reported experiences to provide a broader view of these relationships. A second area concerns technical debt tracking. Future work could test specific tracking mechanisms such as dedicated debt backlogs, issue-tracker labels, and static-analysis dashboards to see whether their effects shift with team size, industry, or codebase age. Using generative-AI technologies to find and fix technical debt is a third field. If human oversight is maintained, more research should be done to determine whether these tools enhance current TDM practices or whether their use creates new types of technological debt and disclosure risk. Future studies should do more than only

record the existence of technical debt. Teams must pay more attention to how they detect, monitor, prioritise, and deal with debt over time. This would provide a clearer basis for understanding which management practices support software quality while also maintaining a positive developer experience.

H. References

[1] R. Verdecchia, P. Kruchten, P. Lago, and I. Malavolta, "Building and evaluating a theory of architectural technical debt in software-intensive systems," Journal of Systems and Software, vol. 176, Jun. 2021, doi: 10.1016/j.jss.2021.110925.

[2] C. Becker, F. Fagerholm, R. Mohanani, and A. Chatzigeorgiou, "Temporal discounting in technical debt: How do software practitioners discount the future?," in Proceedings - 2019 IEEE/ACM International Conference on Technical Debt, TechDebt 2019, Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 23–32. doi: 10.1109/TechDebt.2019.00011.

[3] T. Besker, H. Ghanbari, A. Martini, and J. Bosch, "The influence of Technical Debt on software developer morale," Journal of Systems and Software, vol. 167, Sep. 2020, doi: 10.1016/j.jss.2020.110586.

[4] T. Vhahangwele, B. Isong, and A. A. Mahfouz, "The Impact of Team Dynamics on Software Quality and Productivity: Evidence from South Africa," Journal of Information Systems and Informatics, vol. 8, no. 1, pp. 809–840, Mar. 2026, doi: 10.63158/journalisi.v8i1.1438.

[5] M. O. Ahmad and T. Gustavsson, "The Pandora's box of social, process, and people debts in software engineering," Journal of Software: Evolution and Process, vol. 36, no. 2, Feb. 2024, doi: 10.1002/smr.2516.

[6] H. Ghanbari, S. Ahuja, B. Lee, and J. Gaskin, "Who should pay for technical debt? Exploring software professionals perceptions about technical debt accountability," Information and Organization, vol. 35, no. 3, Sep. 2025, doi: 10.1016/j.infoandorg.2025.100589.

[7] S. Freire et al., "Software practitioners' point of view on technical debt payment," Journal of Systems and Software, vol. 196, Feb. 2023, doi: 10.1016/j.jss.2022.111554.

[8] Z. Li et al., "Technical Debt Management in OSS Projects: An Empirical Study on GitHub," Dec. 2022, [Online]. Available: http://arxiv.org/abs/2212.05537

[9] Xin Wang, Hao Wu, Xiao Liu, Xiao Chen, Li LI, and Jin Liu, "Detecting and Explaining Self-Admitted Technical Debts with Attention-based Neural Networks," in 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE Computer Society, 2020.

[10] H. Gu, S. Zhang, Q. Huang, Z. Liao, J. Liu, and D. Lo, "Self-Admitted Technical Debts Identification: How Far Are We?," in Proceedings - 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2024, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 804–815. doi: 10.1109/SANER60148.2024.00087.

[11] U. M. Graetsch, R. Hoda, H. Khalajzadeh, M. Shahin, and J. Grundy, "Managing technical debt in a multidisciplinary data intensive software team: An observational case study," Journal of Systems and Software, vol. 230, Dec. 2025, doi: 10.1016/j.jss.2025.112546.

[12] M. Wiese and K. Borowa, “IT managers’ perspective on Technical Debt Management,” Journal of Systems and Software, vol. 202, Aug. 2023, doi: 10.1016/j.jss.2023.111700.
[13] D. Albuquerque, E. Guimaraes, G. Tonin, M. Perkusich, H. Almeida, and A. Perkusich, “Comprehending the Use of Intelligent Techniques to Support Technical Debt Management,” in Proceedings - International Conference on Technical Debt 2022, TechDebt 2022, Institute of Electrical and Electronics Engineers Inc., Aug. 2024, pp. 21–30. doi: 10.1145/3524843.3528097.
[14] S. Gosu, D. Feitosa, and P. Avgeriou, “The use of Generative AI in technical debt management: A systematic literature review,” 2026, doi: 10.5281/zenodo.1.
[15] D. Tsoukalas, D. Kehagias, M. Siavvas, and A. Chatzigeorgiou, “Technical debt forecasting: An empirical study on open-source repositories,” Journal of Systems and Software, vol. 170, Dec. 2020, doi: 10.1016/j.jss.2020.110777.
[16] T. B. S. Gomes, D. A. de M. Loiola, and A. R. Santos, “Technical Debt Tools: a Survey and an Empirical Evaluation,” Journal of Software Engineering Research and Development, vol. 12, no. 1, Aug. 2024, doi: 10.5753/jserd.2024.3591.
[17] G. Satya Mohan Chowdary and R. Prasanna Kumar, “Automated Identification and Prioritization of Self-Admitted Technical Debt Using NLP Word Embeddings,” in International Conference on Self Sustainable Artificial Intelligence Systems, ICSSAS 2023 - Proceedings, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 963–971. doi: 10.1109/ICSSAS57918.2023.10331839.
[18] D. Tsoukalas, A. Chatzigeorgiou, A. Ampatzoglou, N. Mittas, and D. Kehagias, “TD Classifier: Automatic Identification of Java Classes with High Technical Debt,” in Proceedings - International Conference on Technical Debt 2022, TechDebt 2022, Institute of Electrical and Electronics Engineers Inc., Aug. 2024, pp. 76–80. doi: 10.1145/3524843.3528094.
[19] D. Albuquerque et al., “Managing Technical Debt Using Intelligent Techniques - A Systematic Mapping Study,” IEEE Transactions on Software Engineering, vol. 49, no. 4, pp. 2202–2220, Apr. 2023, doi: 10.1109/TSE.2022.3214764.
[20] A. Martini, T. Besker, and J. Bosch, “Technical Debt tracking: Current state of practice,” Sci. Comput. Program., vol. 163, pp. 42–61, Oct. 2018, doi: 10.1016/j.scico.2018.03.007.
[21] N. Nikolaidis, D. Zisis, A. Ampatzoglou, A. Chatzigeorgiou, and D. Soudris, “Experience with Managing Technical Debt in Scientific Software Development Using the EXA2PRO Framework,” IEEE Access, vol. 9, pp. 72524–72534, 2021, doi: 10.1109/ACCESS.2021.3079271.
[22] S. Pandya, “AI-Augmented Technical Debt Management in Legacy Software Systems,” in Conference Proceedings - IEEE SOUTHEASTCON, Institute of Electrical and Electronics Engineers Inc., 2026. doi: 10.1109/SoutheastCon63549.2026.11476061.
[23] J. P. Biazotto, D. Feitosa, P. Avgeriou, and E. Y. Nakagawa, “Understanding practitioners’ reasoning and requirements for efficient tool support in technical debt management,” Empir. Softw. Eng., vol. 30, no. 5, p. 134, Sep. 2025, doi: 10.1007/s10664-025-10691-5.

[24] Y. Liu, R. Widyasari, Y. Zhao, I. C. Irsan, J. Chen, and D. Lo, "Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild," Apr. 2026, [Online]. Available: http://arxiv.org/abs/2603.28592

[25] O. Khanvilkar, M. W. Mkaouer, E. A. Alomar, A. Elsaid, A. Chaaben, and M. Touati, "Automated Identification of Machine Learning Technical Debt Code Comments," in Proceedings - 2025 IEEE International Conference on Emerging Technologies and Computing, IC_ETC 2025, Institute of Electrical and Electronics Engineers Inc., 2025. doi: 10.1109/IC_ETC65981.2025.11141122.

[26] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, "An exploratory study on the introduction and removal of different types of technical debt in deep learning frameworks," Empir. Softw. Eng., vol. 26, no. 2, Mar. 2021, doi: 10.1007/s10664-020-09917-5.

[27] E. I. O. Simon, "Characterising Algorithm Debt in Machine and Deep Learning Systems," in Proceedings - International Conference on Software Engineering, IEEE Computer Society, 2025, pp. 216–217. doi: 10.1109/ICSE-Companion66252.2025.00066.

[28] E. L. Melin and N. U. Eisty, "Exploring the Advances in Using Machine Learning to Identify Technical Debt and Self-Admitted Technical Debt," in 2025 IEEE/ACIS 23rd International Conference on Software Engineering Research, Management and Applications, SERA 2025 - Proceedings, Institute of Electrical and Electronics Engineers Inc., 2025, pp. 15–22. doi: 10.1109/SERA65747.2025.11154577.

[29] R. Halepmollasi, "A Composed Technical Debt Identification Methodology to Predict Software Vulnerabilities," in Proceedings - 2020 ACM/IEEE 42nd International Conference on Software Engineering: Companion, ICSE-Companion 2020, Institute of Electrical and Electronics Engineers Inc., Oct. 2020, pp. 186–189. doi: 10.1145/3377812.3381396.

[30] N. E. D. Ferreyra, M. Shahin, M. Zahedi, S. Quadri, and R. Scandariato, "What Can Self-Admitted Technical Debt Tell Us About Security? A Mixed-Methods Study," in Proceedings - 2024 IEEE/ACM 21st International Conference on Mining Software Repositories, MSR 2024, Institute of Electrical and Electronics Engineers Inc., Jul. 2024, pp. 704–715. doi: 10.1145/3643991.3644909.

[31] D. Tsoukalas, N. Mittas, E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and D. Kechagias, "Local and Global Explainability for Technical Debt Identification," IEEE Transactions on Software Engineering, vol. 50, no. 8, pp. 2110–2123, 2024, doi: 10.1109/TSE.2024.3422427.

[32] A. Bhatia, F. Khomh, B. Adams, and A. Hassan, "An Empirical Study of Self-Admitted Technical Debt in Machine Learning Software," ACM Transactions on Software Engineering and Methodology, vol. 35, no. 8, pp. 1–41, Aug. 2026, doi: 10.1145/3785001.

[33] M. Kassab, I. Sohail, and J. P. Fernandes, "The Emotional Cost of Technical Debt: Quantifying Developer Frustration Across Debt-Related Concern Themes," in Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering, New York, NY, USA: ACM, Jul. 2026, pp. 1643–1651. doi: 10.1145/3803437.3805542.

[34] A. Aldaeej and C. Seaman, "A rule-based decision model to support technical debt decisions: A multiple case study of web and mobile app startups," Inf. Softw. Technol., vol. 175, Nov. 2024, doi: 10.1016/j.infsof.2024.107542.