Technical Debt Measurement: An Exploratory Literature Review

Donatien Koulla Moulla [1], Ernest Mnkandla [1], Hayatou Oumarou [2] and Thomas Fehlmann [3]

[1] *University of South Africa, The Science Campus, Florida, 1710, South Africa*
[2] *University of Maroua, Maroua, P.O. Box 46, Cameroon*
[3] *Euro Project Office, Giblenstrasse 50, 8049 Zürich, Switzerland*

Abstract

Measuring Technical Debt is important in guiding software development teams to make informed decisions and prioritize refactoring initiatives. This study presents an exploratory literature review of studies published between 2010 and 2023 to investigate the current state of Technical Debt measurement research. Through a set of four research questions, this study identifies the prevalent methodologies, metrics, and obstacles entailed in quantifying Technical Debt. Specifically, this study focuses on what is proposed to be measured through Technical Debt, the measurement solutions proposed for measuring Technical Debt, and how these approaches categorize and evaluate various aspects of Technical Debt. By scrutinizing the diverse approaches and challenges, this exploratory literature review identifies gaps (and related issues) in Technical Debt measurement research and contributes to a nuanced understanding of Technical Debt measurement practices, offering insights into enhancing software sustainability and maintainability.

Keywords

Technical debt measurement, Technical debt quantification, Technical debt identification, Defect density, Maintainability, Exploratory Literature Review.

1. Introduction and Background

The Technical Debt (TD) metaphor was first introduced by Ward Cunningham in 1992 [1] in the following way: "*Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise*".

Since then, the definition and understanding of TD has evolved [2, 3, 4]. Technical debt has recently gained traction in both industry and the research community. As software systems evolve and grow, managing TD becomes important to ensure long-term sustainability and maintainability of the codebase. However, for TD to be managed, it must be identified and measured [5].

Measuring TD is important for software organizations to understand the extent of the problem and prioritize areas that require immediate attention. By quantifying TD, organizations can make informed decisions about when and where to invest resources to address the accumulated debt [6, 7, 8]. Some studies have highlighted the significant impact of TD on software quality, productivity, and overall project success [9, 10].

Researchers have explored various approaches to quantify TD, and the widely adopted approach involves leveraging code analysis tools that evaluate the quality of the codebase by employing predefined metrics such as code complexity, code duplication, and coding style violations [11, 12]. An alternative approach relies on subjective assessments provided by experienced developers, who can provide insights into TD based on their understanding of the codebase and development process [13].

IWSM-Mensura, September 30 – October 04, 2024, Montréal, Canada
EMAIL: moulldk@unisa.ac.za; mnkane@unisa.ac.za; hayatououmarou@gmail.com; thomas.fehlmann@e-p-o.com
ORCID: [orcid.org/0000-0001-6594-8378]


The most common approach for measuring TD involves various approaches that focus on the different aspects of TD quantification. These approaches include identifying smells, quantifying the Return on Investment (ROI) of refactoring, comparing the ideal state with the current state of software quality, and evaluating alternative development paths to reduce technical debt [14]. However, the lack of consistency among existing tools at the approach and ruleset levels has made it challenging to compare and evaluate these different measurement approaches effectively [15, 16]. To address this issue, a conceptual model called the Technical Debt Quantification Model (TDQM) was developed, which captures key concepts related to technical debt quantification and allows for comparisons and evaluations between different approaches [17].

Except for the literature review of TD in requirements [12], there is a lack of comprehensive and up-to-date reviews synthesizing state-of-the-art techniques and approaches for measuring technical debt. This exploratory literature review examines the current state of research on technical debt measurement. By understanding the extent and impact of technical debt, organizations can make informed decisions about when and where to invest resources to address it.

The remainder of this paper is organized as follows. Section 2 presents the systematic literature review method used in this study. Section 3 presents and discusses the results. Section 4 concludes the paper with a summary of key findings and directions for future work.

2. Review Method

The following section describes the review method used for conducting this exploratory literature review through a Systematic Literature Review (SLR), a rigorous and transparent method for comprehensively analyzing existing research on a specific topic [18]. This study provides a comprehensive overview of existing research on technical debt measurement and identifies research gaps in existing studies, which are used to highlight future research directions. To ensure methodological rigor and transparency, this study followed the SLR guidelines proposed in [18] and adhered to the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) framework [19].

2.1. Research questions

To delve deeper into technical debt measurement, this exploratory literature review addresses the research questions (RQs) presented in Table 1, along with their rationales.

Table 1
Research questions

ID	Research questions	Rationale
RQ1	Why, how, and what should be measured about technical debt (TD)?	To know (identify) what the measurement goals are.
RQ2	What are the existing measurement solutions for measuring technical debt (TD)?	To identify measurement solutions that are available in the literature for measuring technical debt.
RQ3	How do these approaches categorize and evaluate various aspects of TD?	To classify the existing measurement solutions based on various aspects of technical debt.
RQ4	What are the gaps (issues) identified in TD measurement research?	To identify gaps where there are no measurement solutions available yet, and highlight future research directions

2.2. Search strategy

To identify relevant literature to this exploratory review, we conducted a comprehensive search across five reputable digital libraries: Scopus, ScienceDirect, ACM Digital Library, IEEE Xplore, and SpringerLink. The search encompassed peer-reviewed publications published between January 2010 and December 2023. The choice of 2010 as the starting point for the exploratory literature review on technical debt measurement, instead of 1992 when the technical debt metaphor was introduced, is primarily due to the significant increase in research activity around technical debt in the last decade. This study focuses on this period to capture the most recent and relevant developments in the field. Additionally, the selection of this time frame helps ensure that the review includes comprehensive and up-to-date techniques and approaches, reflecting the current state of technical debt measurement practices. These specific databases were chosen because of their extensive coverage of computer science and engineering research, ensuring a high likelihood of capturing pertinent studies on technical debt measurements within this domain.

The search string was based on the key terms identified in the RQs as well as the commonly used terminology associated with TD and measurement. Initially, the main search terms were combined using the Boolean operator "OR" to their corresponding related keywords. Subsequently, these main terms were linked with one another using "AND" to ensure a focused and relevant set of results. Table 2 lists the complete search strings used in this study.

Table 2
Search string

Scope	Search terms
Technical debt	("technical debt" OR "code debt" OR "design debt" OR "architecture debt")
AND Measurement	(measurement OR metrics OR quantification OR evaluation OR siz*)
AND Limitation	(limit* OR gap* OR issue* OR challenges OR weaknesses OR strengths)
AND Classification	(classif* OR categoriz* OR management OR maintainability OR reusability OR testability)
AND Trends	("new approach" OR "emerging technique")

Owing to potential variations in search engine syntax across different databases, we carefully adapted our search strings to optimize retrieval across each database (Scopus, ScienceDirect, ACM Digital Library, IEEE Xplore, and SpringerLink). The search focused on titles, abstracts, and keywords to ensure relevant studies were captured. To manage the retrieval process, we conducted separate searches on each database, followed by consolidation of the identified papers. Subsequently, we employed EndNote reference management software to identify and remove duplicate studies, ensuring streamlined and non-duplicate studies for further analysis.

2.3. Study selection

The selection aimed to identify relevant primary studies that addressed the measurement of technical debt using the following inclusion criteria (IC) and exclusion criteria (EC):

Inclusion criteria:

- Studies written in English and published between January 2010 and December 2023.
- Studies published in peer-reviewed journals, conference proceedings, workshop proceedings, and book chapters.
- Studies where full texts are available.
- Studies that propose, evaluate, or discuss techniques, metrics, or approaches for measuring technical debt.

Exclusion criteria:

- Duplicate publications of the same study.
- Studies where full texts are not available.
- Studies titles and abstracts with a focus on non-technical debt aspects of software development.
- Studies on TD but not on TD measurement.
- Studies on TD without a clear focus on measurement aspects
- Studies have focused solely on identifying TD (without measurement) or lacking details on the measurement approach.

A three-stage process was followed to select the studies for this review.

In the first stage, 808 primary studies were identified from the five digital libraries. Seventy (70) studies were discarded as duplicate publications of the same study.

In the second stage, an initial screening of the search results (738 studies) was performed based on titles and abstracts. Studies that full texts are not available were excluded (228 studies). Additionally, 480 studies that did not explicitly mention technical debt measurement or quantification techniques, metrics, or approaches were excluded. After this stage, we obtained 30 full-text studies reviewed.

In the third stage, a full-text review of the remaining studies (30 remaining studies) was conducted. During this phase, we applied the defined inclusion and exclusion criteria to ensure the relevance and quality of the selected studies. The exclusion criteria were as follows:

- Studies on TD but not on TD measurement.
- Studies on TD without a clear focus on measurement aspects.
- Studies have focused solely on identifying TD (without measurement) or lacking details on the measurement approach.

To ensure the reliability of the study selection process, two researchers independently performed screening and full-text reviews. Any disagreements or conflicts were resolved by discussion and consensus. After this phase, the 21 remaining studies were included in the review. The study selection process, with the total number of studies retrieved and included in each phase, is shown in Figure 1.

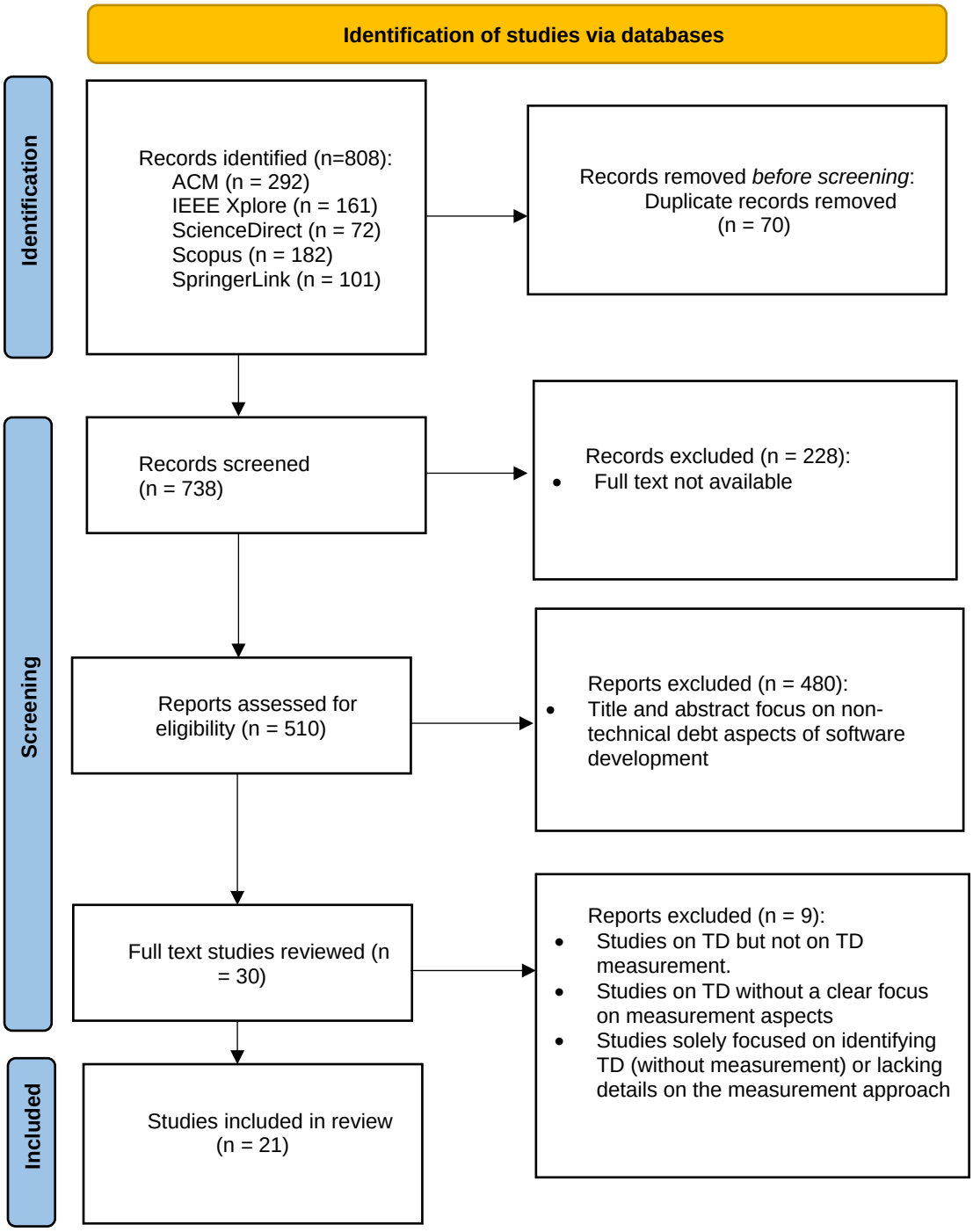

Figure 1: PRISMA flow diagram of the literature selection process

2.4. Quality assessment

Quality assessment was used to assess the relevance and credibility of the selected studies. The papers were selected from five well-known databases of papers reviewed by experts before their publication. The quality of the included primary studies was assessed using a customized quality

checklist adapted from Kitchenham and Charters [18] that was the most appropriate for our research questions.

The checklist consists of the following four quality criteria:

- QA1: Are the research aims clearly stated?
- QA2: Is the technical debt measurement approach clearly described?
- QA3: Are the findings clearly stated?
- QA4: Are the limitations of the study discussed?

Only studies that answered at least three of the above questions were selected. After this phase, all the 21 studies met three of the above questions.

2.5. Data extraction and Synthesis

A data extraction form was developed to retrieve relevant information from the included primary studies, addressing the RQs. It should be noted that not all the selected studies addressed all four RQs. Data synthesis aimed to collate and summarize the results of the included primary studies. We identified and grouped all relevant data to answer the RQs using a descriptive synthesis.

3. Results and Discussion

This section provides answers to the SLR research questions based on a synthesis of the selected studies.

3.1. RQ1: Why, how, and what should be measured about technical debt (TD)?

The scope of the RQ1 covers a triple question “why”, “how” and “what”. Why measuring TD enables to understanding the reasons behind measuring technical debt helps in recognizing its importance and the benefits it brings to software projects. How to Measure TD focuses on the methodologies and approaches used to quantify and analyze TD. What Should be Measured about TD identifies the specific aspects and metrics that need to be quantified to understand and manage technical debt effectively.

The primary measurement goals identified in the selected studies are:

1. Sizing TD in terms of the effort required to reduce it to zero (S1, S3, S4, S5, S8, S19, S20, S21).
2. Assessing the impact of TD on software quality attributes such as functionality, performance, maintainability, reliability, and security (S1, S2, S6, S9, S10, S17, S18).
3. Estimating the rework/refactoring efforts required to enhance evolvability and mitigate accumulated TD (S3).
4. Evaluating the accuracy and usefulness of TD measurement tools (S11).
5. Comparing different TD identification techniques (S13).

Several approaches have been proposed for measuring (quantifying) TD. Existing approaches measure TD in terms of what is proposed to be measured through TD in the following ways:

- Eight studies (S1, S3, S4, S5, S8, S19, S20, S21) base their quantification on the identification of code, design, architectural smells, or defects.
- Seven studies (S3, S4, S5, S8, S19, S20, S21) have attempted to quantify the return on investment (ROI) of refactoring activities to remove technical debt.
- Seven studies (S1, S2, S6, S9, S10, S17, S18) compare an ideal state with the current state of the software in terms of quality attributes, such as maintainability and modularity.
- Seven studies (S3, S4, S5, S8, S19, S20, S21) compare alternative development paths with the aim of reducing rework and quantifying the impacts of taking on technical debt versus not taking it on.

In summary, the key measurements relate to sizing TD in terms of the required remediation effort, assessing the impact on software quality attributes, estimating rework efforts for mitigating TD, and evaluating the effectiveness of TD measurement approaches and tools.

3.2. RQ2: What are the existing measurement solutions for measuring technical debt (TD)?

The selected studies proposed various measurement solutions, including:

- Code metrics (complexity, coupling, cohesion, size, duplication, etc.) (S1, S3, S6, S9, S10, S11, S14, S16, S18).
- Modularity and design quality metrics (S3, S14).
- Defect proneness, change proneness, and maintenance effort metrics (S7, S9, S10, S15, S17).
- Static code analysis issues/violations (S2, S9, S19, and S20).
- Test coverage and quality metrics (S2, S9).
- Technical debt principal and interest calculations (S5, S7, S12, S15, S19, S20, and S21).
- Machine Learning models for TD identification and forecasting (S6, S9, and S10).

Table 3 presents the measurement solutions/metrics proposed for measuring Technical Debt (TD) across the selected studies.

Table 3
TD measurements / metrics

Types of TD measurement / metrics	TD Measurements / metrics
Code Metrics	• Number of parameters, comments, expression statements, variable declarations, name expressions, loop statements, assignment statements, math operations, string literals, number of literal (S1) • Lines of code (LOC) (S1, S11, S16) • Number of methods calls (coupling) (S1) • Cyclomatic Complexity (CC) (S11, S16) • S101 Fat, S101 XS (complexity measures) (S11) • Coupling Between Object (CBO) (S6, S10) • Weight Method Count (WMC) (S6, S10, S18) • Depth of Inheritance Tree (DIT) (S6, S10, S16) • Response for Class (RFC) (S6, S10) • Lack of Cohesion in Methods (LCOM) (S6, S16, S18) • Number of Static Invocations (NOSI) (S6) • Duplicated Lines Density (S10) • Comment Lines Density (S10)
Design/Architecture Metrics	• Complexity Index (Path, Activity, Application level) (S3) • Modularity Index (Path, Activity, Application level) (S3) • Data Coupling Index (Path, Activity, Application level) (S3) • Coupling metrics (MPC, MOA) (S14) • Cohesion metric (LCOM) (S14) • Polymorphism metric (NOP) (S14) • Inheritance metric (DIT) (S14) • Size metrics (CIS, SIZE1/LOC) (S14)
Static Analysis Metrics	• Number of static code analysis issues/violations (S2) • Sqale_index, Reliability_remediation_effort, Security_remediation_effort, Total_principal (S9)

Quality Metrics	• Bugs, Code_smells, Vulnerabilities, Complexity, Uncovered_lines, Duplicated_blocks (S9) • Reliability (measured by bug statistics) (S2) • Testability (measured by test coverage) (S2) • Extensibility (variation in estimations for similar user stories) (S2) • Exchangeability (measured by release effort over time) (S2) • Defect Proneness (DP), Maximum Defects per 100 LOC Touched (MaxDP), Extra Defect Proneness (EDP), Maximum Extra Defects per 100 LOC (MaxEDP), Relative Extra Defect Proneness (REDP), Average Relative Extra Defect Proneness (AREDP), Violation Density (VD), Linkage, Refactoring_index (S7, S15)
Effort/Cost Metrics	• Efforts Deviation Index (S3) • Repair Effort, Rework Fraction, Rebuild Value, Refactoring Adjustment (S21) • Maintenance Effort, Maintenance Fraction, Quality Factor (S21) • Refactoring cost, Investigation cost, Modification cost, Workaround cost, Customer support cost, Patch cost, Validation cost, Maintenance cost, Cost of delay, Remediation cost, Non-remediation cost, Reengineering cost, Contingent Cost, Implementation Cost, Cost per LOC (S5) • Rework, Revenue, NPV, ROI, Investment's expected net value, Loss of Business (S5)
Machine Learning Features	• Commits_count, Code_churn_avg, Contributors_count, Contributors_experience, Hunks_count, Issue_tracker_issues, Max_nested_blocks, Total_methods, Total_variables, Total_refactorings (S10)
Other Metrics	• Development velocity (S2) • LOC maintained between versions (estimate of future maintenance load) (S7) • Afferent Couplings (AC), Efferent Couplings (EC), Number of children, Code duplication, Documentation related measures (S16) • Modularity violations, Grime, Code smells, ASA issues, Size, Defect-proneness, Change-proneness (S17) • Abstractness (Abstr), Average Line of Code per Method (ALCM), Distance from Main Sequence (DMS), Weighted Method Count (WMC), Average Method Weight (AMW), Changing Classes (CC), Number of Called Classes (FANOUT), Access to Foreign Data (ATFD), Locality of Attribute Accesses (LAA), Tight Class Cohesion (TCC) (S18)

As shown in Table 3, the measurement solutions/metrics proposed across the selected studies covered a wide range of aspects, including code metrics, design/architecture metrics, static analysis metrics, quality metrics, effort/cost metrics, machine learning features, and other metrics related to defect proneness, change proneness, and maintainability.

3.3. RQ3: How do these approaches categorize and evaluate various aspects of TD?

The selected studies categorize and evaluate various aspects of TD in the following ways:

- Based on the type of TD: code debt, design debt, architectural debt, documentation debt, test debt, etc. (S3, S4, S10, S12, S18, S19, and S20).
- Based on architectural levels: path, activity, and application levels (S3).
- Based on quality characteristics/attributes affected: reliability, security, maintainability, portability, etc. (S4, S13, S19, and S20).
- Based on severity/priority of issues (S1, S19, S20).
- Based on financial cost/effort estimations (S5, S19, S20, and S21).

In summary, the approaches categorize and evaluate TD based on numerous factors, including the types of TD (code, design, architectural, etc.), architectural levels affected, quality characteristics impacted, severity or priority of issues, and financial implications or effort estimations. By considering these different categories and evaluation perspectives, this study aims to provide an understanding of TD, its manifestations, its impact on various aspects of software quality, and the potential costs and efforts required for its remediation.

3.4. RQ4: What are the gaps (issues) identified in TD measurement research?

From the analysis of selected studies, gaps (and related issues) in TD measurement research were identified by the researchers themselves:

- Lack of validation on real-world projects (S3).
- Measuring other types of debt beyond code debt (documentation debt, test debt, architectural debt, etc.) (S4, S10).
- Integrating developers' opinions with code characteristics to improve TD severity identification (S1).
- Holistic approaches combining financial and technical factors (S14).
- Interpretable thresholds in metric-based approaches (S14).
- Quantifying interest costs, risks/liabilities, and opportunity costs of TD (S19 and S20).
- Generalizability and application to complex systems (S21).
- Need for effective tooling and methodologies for managing TD across the software development lifecycle (S10).

These gaps and issues highlight the need for further research and improvements in TD measurement, including validation on real-world projects, comprehensive coverage of different TD types, integration of developer perspectives, holistic approaches, interpretable thresholds, consideration of TD costs and risks, generalizability to complex systems, effective tooling and methodologies, direct TD quantification, and accounting for non-functional requirements.

Addressing these gaps and issues can contribute to more accurate, practical, and comprehensive TD measurement approaches, thereby enabling better management and decision-making processes related to TD in software development projects.

3.5. Discussion

This study presents several key findings. Firstly, the study identified a variety of measurement goals such as sizing TD, assessing its impact on software quality attributes, estimating rework efforts for TD mitigation, and evaluating the effectiveness of TD measurement tools. The research highlights the prevalent methodologies, including code metrics, design/architecture metrics, static analysis metrics, quality metrics, effort/cost metrics, and machine learning features for identifying and forecasting TD.

3.5.1. Implications for Researchers

The findings indicate significant research gaps and areas for further exploration. Researchers are encouraged to focus on:

1. Validation in Real-World Projects: There is a need for more empirical validation of TD measurement tools and methodologies in real-world software development projects to enhance their practical applicability.
2. Comprehensive Coverage of TD Types: Future research should expand beyond code debt to include other types such as documentation debt, test debt, and architectural debt. This holistic approach will provide a more accurate picture of TD and its implications.
3. Integration of Developer Perspectives: Incorporating insights from developers regarding the severity and impact of TD can improve the accuracy and relevance of measurement tools.
4. Holistic Approaches Combining Financial and Technical Factors: Developing measurement approaches that consider both financial and technical aspects of TD can offer more comprehensive management strategies.
5. Interpretable Thresholds in Metric-Based Approaches: Establishing clear and interpretable thresholds for various TD metrics will facilitate better decision-making processes for software teams.

3.5.2. Implications for Practitioners

Practitioners can benefit from the study's insights by:

1. Adopting Diverse Measurement Solutions: Utilizing a combination of code metrics, design/architecture metrics, static analysis metrics, and machine learning models can provide a multifaceted understanding of TD, aiding in more effective management and reduction strategies.
2. Focusing on High-Impact Areas: By identifying and prioritizing areas with high TD, practitioners can allocate resources more efficiently, addressing the most critical issues that affect software quality and maintainability.
3. Continuous Monitoring and Refinement: Implementing continuous TD measurement and monitoring processes will help in early detection and mitigation of TD, thereby reducing long-term costs and improving software sustainability.

3.5.3. High-Level Concepts and Lessons Learned

From the synthesis of over a decade of research, several high-level concepts and lessons emerge:

1. The Complexity of TD Measurement: The measurement of TD is inherently complex, involving multiple dimensions such as code quality, design, architecture, and financial implications. This complexity necessitates sophisticated and integrated measurement approaches.
2. The Importance of Contextual Factors: The impact of TD varies significantly depending on the context of the software project, including factors like project size, complexity, and team expertise. Tailoring TD measurement approaches to specific project contexts can enhance their effectiveness.
3. Need for Standardization and Tool Integration: There is a pressing need for standardization in TD measurement approaches and better integration of tools to facilitate more consistent and reliable measurements across different projects and organizations.
4. Continuous Evolution of Measurement Techniques: As software development practices evolve, so too must the techniques and tools for measuring TD. Keeping abreast of emerging trends and incorporating new methodologies will be crucial for maintaining effective TD management practices.

In summary, addressing these insights and integrating them into both research and practice can significantly enhance the management of TD, contributing to the long-term sustainability and quality of software systems.

4. Threat to Validity

This exploratory literature review aimed to provide a comprehensive overview of state-of-the-art in technical debt measurement research. However, there are potential threats to validity that should be acknowledged:

- While the search string was carefully constructed to capture relevant studies, it is possible that some relevant publications may have been missed because of the use of different terminologies or the presence of relevant studies in sources not included in the selected digital libraries. This review focused exclusively on studies published between January 2010 and December 2023. Consequently, relevant earlier works or very recent publications may have been unintentionally excluded.
- Although the study selection process followed well-defined inclusion and exclusion criteria and was conducted independently by two researchers, there is an inherent risk of bias in the selection and interpretation of studies.
- The quality assessment of the included studies was based on a customized checklist adapted from established guidelines. However, the assessment process may have introduced bias because of the subjective interpretation of quality criteria.
- The included studies exhibited substantial heterogeneity in terms of research methodologies, measurement approaches, and evaluation contexts. This diversity may introduce challenges in synthesizing and comparing findings across studies.

Despite these potential threats, we used rigorous and systematic methods to conduct the literature review, including following established guidelines involving multiple researchers in the study selection and data extraction processes. Additionally, we have transparently acknowledged the limitations and potential threats to validity, which can inform the interpretation and applicability of the findings.

5. Conclusion and Future Work

5.1. Summary of findings

The quantification and measurement of TD are important for software development teams and organizations. By understanding the extent and impact of technical debt, organizations can make informed decisions about when and where to invest resources to address it. A number of technical debt issues have been investigated by researchers over the years, but relatively few have focused on technical debt measurements/metrics. This 2010-2023 SLR in studies proposing TD measurements followed the guidelines proposed by Kitchenham and Charters [18] and adhered to the PRISMA framework [19]. 21 studies included in the review were selected from the Scopus, ScienceDirect, ACM Digital Library, IEEE Xplore, and SpringerLink digital libraries to address our research questions using specified inclusion and exclusion criteria, and then analyzed to answer the research questions.

The key findings are as follows:

- RQ1: Why, how, and what should be measured about technical debt (TD)?

The primary measurement goals identified were:

1) sizing TD in terms of required remediation effort,
2) assessing the impact of TD on software quality attributes,
3) estimating rework efforts for TD mitigation,
4) evaluating the effectiveness of TD measurement approaches and tools.

- RQ2: What are the existing measurement solutions for measuring TD?

A wide range of measurement solutions has been proposed, including code metrics (complexity, coupling, cohesion, etc.), design and architecture metrics, static analysis metrics, quality metrics, effort/cost metrics, machine learning features, and metrics related to defect proneness, change proneness, and maintainability.

- RQ3: How do these approaches categorize and evaluate various aspects of TD?

The approaches categorized and evaluated TD based on factors such as the types of TD (code, design, architectural, etc.), architectural levels affected, quality characteristics impacted, severity or priority of issues, and financial implications or effort estimations.

- RQ4: What are the gaps (issues) identified in TD measurement research?

The key gaps identified included the lack of real-world validation, limited coverage of non-code debt types, need for holistic approaches integrating technical and financial factors, lack of interpretable thresholds, quantification of TD costs and risks, generalizability to complex systems, and the need for effective tooling and methodologies.

In summary, this review identified a diversity of measurement solutions and categorization approaches for technical debt while also highlighting significant gaps and areas for further research and improvement in this field. The findings provide a nuanced understanding of the current state of technical debt measurement research and offer insights into enhancing software sustainability and maintainability through effective technical debt management practices.

5.2. Future Work

The findings from this exploratory literature review highlight promising directions for future research on technical debt measurements. There is a need for more comprehensive measurement approaches that can effectively quantify and consolidate distinct types of technical debt such as design debt, architectural debt, documentation debt, and test debt. Further research could also explore the identification and measurement of technical debt, which can be derived from software functional requirements not yet implemented, as well as from system non-functional requirements not implemented and that can be implemented in software functions distributed across a software environment.

Acknowledgment

We are grateful to anonymous reviewers.

References

[1] W. Cunningham, The WyCash Portfolio Management System, in: Proceedings of the 7th International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA, Association for Computing Machinery, New York, NY, USA, 1992, pp. 29–30. doi:10.1145/157709.157715

[2] I. Gat, (Ed.), Special Issue: Technical Debt, Cutter IT J., vol. 23, no. 10, 2010.

[3] M. Fowler, Technical Debt, blog, 2019. URL: http:// martinfowler.com/bliki/TechnicalDebt.html.

[4] P. Kruchten, R. L. Nord, and I. Ozkaya, Technical debt: From metaphor to theory and practice, IEEE software, 29(6) (2012) 18–21. doi:10.1109/MS.2012.167

[5] N. Rios, M.G.d. Mendonça Neto, and O.R. Spínola, A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners, Information and Software Technology, 102 (2018) 117–145.

[6] Y. Guo, and C. Seaman, A portfolio approach to technical debt management, in: Proceedings of the 2nd Workshop on Managing Technical Debt, Association for Computing Machinery, New York, NY, USA, 2011, pp. 31–34. doi:10.1145/1985362.1985370.

[7] V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, F. A. Fontana, A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools, Journal of Systems and Software, 171 (2011) 1–16. doi:10.1016/j.jss.2020.110827.

[8] C. Seaman, Y. Guo, Chapter 2 - Measuring and Monitoring Technical Debt, in: Marvin V. Zelkowitz (Ed.), Advances in Computers, Elsevier, 82 (2011), pp. 25–46. doi:10.1016/B978-0-12-385512-1.00002-5.

[9] G. Freitas, J. H. Bernardo, G. SizíLio, D. A. Da Costa, and U. Kulesza, Analyzing the Impact of CI Sub-practices on Continuous Code Quality in Open-Source Projects: An Empirical Study, in

Proceedings of the 37th Brazilian Symposium on Software Engineering (SBES '23), Campo Grande, Brazil, 2023, pp. 1–10, 2023. doi:10.1145/3613372.3613403.
[10] F. A. Fontana, R. Roveda, S. Vittori, A. Metelli, S. Saldarini, and F. Mazzei, On evaluating the impact of the refactoring of architectural problems on software quality, in: Proceedings of the Scientific Workshop Proceedings of XP2016 (XP '16 Workshops), Edinburgh Scotland, United Kingdom, 2016, pp. 1–8. doi:10.1145/2962695.2962716.
[11] N. Zazworka, A. Vetro', C. Izurieta, et al., Comparing four approaches for technical debt identification, Software Quality Journal, 22 (3) (2014) 403–426. doi:10.1007/s11219-013-9200-8
[12] A. Melo, R. Fagundes, V. Lenarduzzi, W. B. Santos, Identification and measurement of Requirements Technical Debt in software development: A systematic literature review, Journal of Systems and Software, 194 (2022) 1–21. doi:10.1016/j.jss.2022.111483.
[13] P. Boris, C. Castellanos, D. Correal, et al., Technical debt payment and prevention through the lenses of software architects, Information and Software Technology, 140 (2021) 1–16. doi:10.1016/j.infsof.2021.106692.
[14] U. Vora, Measuring the Technical Debt, in: Proceedings of the 17th Annual System of Systems Engineering Conference (SOSE), Rochester, NY, USA, 2022, pp. 185–189, doi: 10.1109/SOSE55472.2022.9812632.
[15] M. Mathioudaki, D. Tsoukalas, M. Siavvas, and D. Kehagias, Comparing Univariate and Multivariate Time Series Models for Technical Debt Forecasting, in: Proceedings of Computational Science and Its Applications – ICCSA 2022 Workshops, Malaga, Spain, 2022, pp. 62–78. doi:10.1007/978-3-031-10542-5_5.
[16] M. Mathioudaki, D. Tsoukalas, M. Siavvas, and D. Kehagias, Technical Debt Forecasting Based on Deep Learning Techniques, in: Proceedings of Computational Science and Its Applications – ICCSA 2022 Workshops, Cagliari, Italy, 2021, pp. 306–322. doi:10.1007/978-3-030-87007-2_22.
[17] J. Perera, Modelling the Quantification of Technical Debt, in: Companion Proceedings of the 2022 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion 2022), Auckland, New Zealand, 2022, pp. 50–53. doi:10.1145/3563768.3565553.
[18] B. A. Kitchenham, and S. Charters, Guidelines for performing systematic literature review in software engineering, Technical Report, Keele University, 2007.
[19] M. J. Page, J. E. McKenzie, P. M. Bossuyt, I. Boutron, T. C. Hoffmann, C. D. Mulrow et al., The PRISMA 2020 statement: an updated guideline for reporting systematic reviews, BMJ., 2021. doi:10.1136/bmj.n71.

Appendix – selected primary studies.

This appendix contains the supporting documentation for our article. The list of the 21 selected primary studies to perform the SLR is presented in Table 1.

Table 1
Selected primary studies

Study ID	Authors	Title	Source
S1	Dongjin Yu et al.	Identifying the severity of technical debt issues based on semantic and structural information	Software Quality Journal
S2	Markus Finke et al.	How to introduce TD Management into a Software Development Process – A Practical Approach	ACM/IEEE International Conference on Technical Debt (TechDebt)
S3	Urjaswala Vora	Measuring the Technical Debt	Annual System of Systems Engineering Conference (SOSE)
S4	Luka P. et al.	The Gap between the Admitted and the Measured	Applied Sciences

		Technical Debt: An Empirical Study	
S5	Judith Perera	Modelling the Quantification of Technical Debt	Companion Proceedings of the 2022 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion 2022)
S6	Lerina AVERSANO	Forecasting technical debt evolution in software systems: an empirical study	Frontiers of Computer Science
S7	Elvira-Maria Arvanitou et al.	Quantifying TD Interest: Are we Getting Closer, or Not Even That?	Euromicro Conference on Software Engineering and Advanced Applications (SEAA)
S8	Ana Melo et al.	Identification and Measurement of Technical Debt Requirements in Software Development: a Systematic Literature Review	Journal of Systems and Software
S9	Dimitrios Tsoukalas et al.	A Clustering Approach Towards Cross-Project Technical Debt Forecasting	SN Computer Science
S10	Dimitrios Tsoukalas et al.	Machine Learning for Technical Debt Identification	IEEE Transactions on Software Engineering
S11	Jason Lefever et al.	On the Lack of Consensus Among Technical Debt Detection Tools	IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)
S12	Paris Avgeriou et al.	An Overview and Comparison of Technical Debt Measurement Tools	IEEE Software
S13	Peter S. et al.	Comparing Maintainability Index, SIG Method, and SQALE for Technical Debt Identification	Annual ACM Symposium on Applied Computing (SAC '20)
S14	Makrina Viola Kosti et al.	Technical Debt Principal Assessment through Structural Metrics	Euromicro Conference on Software Engineering and Advanced Applications (SEAA)
S15	Davide Falessi and5Andreas Reichel	Towards an Open-Source Tool for Measuring and Visualizing the Interest of Technical Debt	International Workshop on Managing Technical Debt (MTD)
S16	Clauirton A. Siebra et al.	Applying Metrics to Identify and Monitor Technical Debt Items during Software Evolution	IEEE International Symposium on Software Reliability Engineering Workshops
S17	Nico Zazworka et al.	Comparing four approaches for technical debt identification	Software Quality Journal
S18	Francesca Arcelli Fontana et al.	Investigating the Impact of Code Smells Debt on Quality Code Evaluation	International Workshop on Managing Technical Debt (MTD)
S19	Bill Curtis et al.	Estimating the Size, Cost, and Types of Technical Debt	International Workshop on Managing Technical Debt (MTD)
S20	Bill Curtis et al.	Estimating the Principal of an Application’s Technical Debt	IEEE Software
S21	Ariadi Nugroho et al.	An Empirical Model of Technical Debt and Interest	Proceedings of the 2nd Workshop on Managing Technical Debt (MTD '11)