# Revisiting Technical Debt Types and Indicators for Software Systems

Dilek Çağlayan[1,2]
[1]Information Management Directorate
ASELSAN A.S.
Ankara, Türkiye
dilek.caglayan@metu.edu.tr

Özden Özcan-Top[2]
[2]Dept. of Information Systems, Graduate of Informatics
Middle East Technical University
Ankara, Türkiye
ozdenoz@metu.edu.tr

## ABSTRACT

Technical Debt (TD) [1] term in software systems was introduced over two decades ago and remains a critical concern in software development. It has the potential to evolve into a liability that necessitates refactoring or rewriting code over time. Regardless of its significance, there exists a notable gap in literature concerning a comprehensive list of technical debt indicators. The purpose of this study is to re-evaluate existing TD categorization and extend TD indicators and offer a complete and validated TD Type and TD Indicator list. In this study, we adopted a qualitative research approach and used mapping and expert opinion techniques as the research approach. The number of TD indicators extracted from existing formal literature was 60 which was extended to 92 by reviewing gray literature. This list was then subjected to the expert review, and with their feedback, grew by an additional 21%. Consequently, we present 10 distinct TD types, accompanied by 120 TD indicators that would aid in TD identification, resolution and minimizing the risks and costs associated with technical debt in software development.

## KEYWORDS

Technical Debt, Technical Debt Types, Technical Debt Indicators, TD Categorization, TD identification

## 1 INTRODUCTION

The metaphorical term of "technical debt (TD)" was coined in the beginning of the 90's [1]. It was proposed as an analogy by Ward Cunningham, and its occurrence and TD management activities are still being investigated in today's IT world. Technical debt concept emerges from developing a proper product with a "not-quite-right" code intentionally or unintentionally [1]. Even though the product with a non-ideal solution is accepted by the customer, it could turn into a liability issue that needs to be refactored or redeveloped again.

In Agile projects, the fast-paced feedback mechanisms and iterative nature of development makes the occurrence of technical debt highly possible [1]. The iterative nature of Agile methodologies often prioritizes the delivery of functional software. This focus can potentially lead to shortcuts or temporary solutions that accumulate as technical debt. Furthermore, this issue is compounded by the growing complexity and interdependency of modern software systems. These factors present significant challenges in managing technical debt in Agile development.

Similar to how financial debt creates interest, technical debt also builds up *interest* over time. Organizations with accumulated TD are likely to encounter negative impacts in the long term [2]. These include reduced agility, increased maintenance costs, and diminished software quality. This, in turn, can lead to a loss of competitive advantage and decreased customer satisfaction.

Technical debt can take many forms in software systems. To share a common vocabulary with the TD research community, it is important to organize existing knowledge about TD types and TD indicators [3]. TD types refer to categories of misapplications made and shortcuts taken in software systems. TD indicators are measurable attributes or signs that point to the presence or degree of technical debt within a software project [4].

Both TD types and TD indicators help differentiate the root causes of issues; identifying, managing, and resolving different forms of technical debt in software systems.

Although technical debt is readily discussed in the literature [2,3,5,9,10,12,13], there is no consensus on technical debt types and associated indicators in software projects. While such studies have delved into the subject of technical debt, the focus has predominantly been on a narrow range of indicators, often featuring the same few measures in a repetitive manner and the abstraction levels of the TD types were diverged. This inconsistency hinders understanding different facets of technical debt but also restricts the scope for effective management and mitigation strategies.

From this perspective, the primary purpose of this paper is to address this gap by providing a detailed catalog of technical debt indicators associated with various TD types. . To achieve this purpose, we followed a two-staged research approach: (1)



*SAC'24, April 8 –April 12, 2024, Avila, Spain*

DOI: 10.1145/3605098.3636043

specification and mapping of existing TD types and indicators from the literature; (2) revising and extending the existing TD indicators through experts' review.

The rest of this paper is structured as follows: Section 2 outlines the research methodology and highlights the threats to the validity of this research; Section 3 contains the results of the mapping and expert opinion interviews; Sections 4 presents and discusses the results obtained; and finally, Section 5 contains the conclusions and future studies.

## 2 RESEARCH METHODOLOGY

In this study, a qualitative research methodology was preferred to explore technical debt phenomena in software systems [6].

We basically followed a combination of a literature mapping study [7] and expert opinion (i.e. expert judgment) [8] to ensure both breadth and depth of our findings. We preferred the mapping approach for the structured analysis of the diverse TD types and indicators documented in the literature. It also allowed us to group similar indicators together, recognize patterns, and identify inadequately represented or overrepresented categories. Additionally, this approach helped us to identify gaps in the existing literature by mapping out the existing evidence and pinpointing areas that require further exploration. To further validate the initial list derived from the literature, we employed the "Expert Opinion" approach which involved consulting with subject matter experts to confirm, revise, or expand the TD types and indicators identified.

Each step we followed is depicted in Figure 1 and explained in detail below.

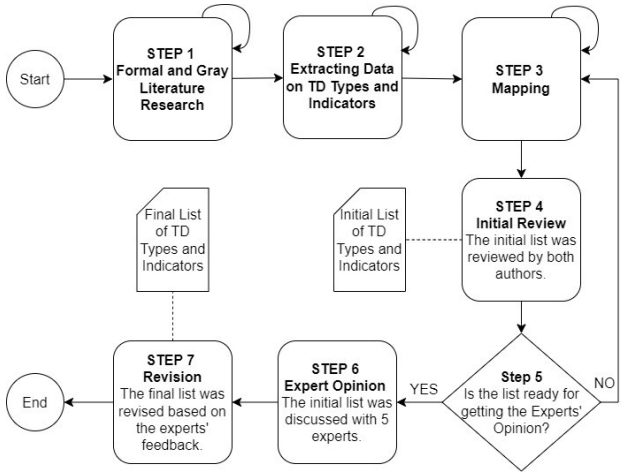


**Figure 1: Followed research steps**

**Step 1: Formal and Gray Literature Research on TD Types and Indicators**

The first step in our methodology was an in-depth literature review focused on identifying various types of technical debt. In this step, as the beginning of the mapping study, we searched for relevant, peer-reviewed studies in ACM Digital Library, SpringerLink, and ScienceDirect databases. The search strings used are "Technical Debt OR TD", "technical debt type* or TD Type*", "technical debt indicator* OR TD indicator*". To ensure a comprehensive scope, we also applied the snowballing technique, checking the references of each selected study to find more potential studies for inclusion. As a result, we selected ten studies published over a ten-year period from 2011-2021. These studies [2,4,5,9,10,11,12,13,15,16] focus on specification or evaluation of TD types/indicators in software systems.

As we aimed for a more complete understanding of TD indicators, our research on TD indicators was supported by the findings from gray literature sources. These sources include internet blog posts, lecture notes, and white papers. In gray literature search, we targeted specific terms that had broad meanings to point out a single indicator. While the first step provided the "what" part (i.e. the types of technical debt), the second step aimed to define the "how" part (i.e. the specific indicators outline the existence of each debt category).

**Step 2 - Step 3: Extracting Data on TD Types and Indicators, and Mapping Process**

***TD Type Specification.*** We determined that the number of TD types vary among research papers. For example, Alves et al. [3] have mentioned 15 distinctive categories in their systematic mapping study; both Lenarduzzi et al. [9] and Li et al. presented 10 distinct categories [5]. Upon examining the gathered data, we have noticed that some TD types overlap with others. For example, "defect debt" may fit under "test debt", and "build debt" could be seen as a subset of "code debt" category. We studied all TD types mentioned in the literature and identified eight distinct categories. These are *Architecture Debt, Code Debt, Design Debt, Documentation Debt, Infrastructure Debt, People Debt, Requirements Debt,* and *Test Debt.* These categories clearly differentiate the various indicators from one another.

However, we also detected gaps in the categorization. To make it more comprehensive, we added two new TD types: *Configuration Debt* and *Management Debt*. This resulted in a refined list of ten distinct technical debt categories.

***TD Indicator Specification and Mapping.*** In this step, first we focused specifically on extracting the indicators associated with each technical debt type from formal literature. The initial list of technical debt indicators that we found in formal literature comprised 60 indicators across 10 technical debt types. We identified 32 TD indicators from gray literature sources, which were highlighted with gray color on the Final TD Indicators List. By the end of Step 2, our list comprised 92 TD indicators linked to the 10 TD types.

Upon collecting data related to the TD indicators both from formal and gray literature, it was observed that certain TD indicators were recurring in multiple sources. To correctly extract information from each study, we explored the meanings of the repeated indicators. To avoid indicator duplication, we combined the ones with similar meanings. Brief explanation of each indicator was added to the initial TD types and indicators list so as to avoid different interpretations of the indicators.

The process specifying TD indicators from the literature was iterated five times until we could not add new TD indicators anymore.

**Step 4: Initial Review**

After completing the mapping study, an initial review was carried out on the compiled list of Technical Debt (TD) types along with their corresponding indicators. In this phase, the authors collaboratively examined the list, and identified certain TD types that were not comprehensively covered by the existing indicators. As a result of the analysis of indicators found, nine more indicators were added to these TD types to achieve a more complete coverage. The output of this step has served as the initial list that outlines various types of TD and corresponding indicators, serving as a basis for further validation.

**Step 5: Decision Point**

This step served as a critical point in the research process in which we assessed whether the initial list was sufficiently comprehensive and robust for the next phase in an iterative way. If the list meets the predefined criteria for completeness and accuracy, the process proceeds to Step 6. If not, it cycles back to Step 3 for revision.

After five cycles, we determined that the list was ready to discuss with the experts, so we moved on to Step 6.

**Step 6: Expert Opinion**

This step started with the selection of the experts. After choosing the experts, the interviews were conducted with each one of them.

***Selection of the Experts***. After finalizing the initial TD list, we focused on selecting experts. We choose five experts with senior or executive-level experience from both academic and industry perspectives to achieve a more holistic understanding of technical debt, its implications, challenges, and strategies for management.

Four experts were chosen from the IT industry, spanning roles from a CTO to senior developers. Their diverse experiences, as shown in Table 1, cover a range of projects including software product development, e-commerce platform development, business process automation, system and website development, mobile app development, and cloud migration and integration. Hence, these professionals offered practical insights into the day-to-day management of technical debt and its impact on various types of projects.

The fifth expert was a professor from academia. His expertise ensured that our study aligns with current academic theories and methodologies related to technical debt. The professor's insights offered a broad, theoretical perspective and deepened our understanding of technical debt's long-term implications.

The combination of a professor and industry professionals allowed for a detailed exploration of technical debt. The CTO and professor offered a macro, strategic viewpoint, aligning technical debt with broader organizational objectives and academic theories. On the other hand, the technical lead and senior developers provided a micro, operational perspective, focusing on immediate challenges and practical solutions in software development. This diverse group ensured that our study encapsulated both the high-level strategic implications and the ground-level operational challenges of technical debt, leading to a balanced integration of academic and real-world practices.

***Conducting Interviews with the Experts***. Before the interviews we developed a spreadsheet including an introduction page detailing the study's general information and the *TD Types and Indicators List* page, which lists the 10 technical debt types and associated 101 indicators. Another section, the *Expert Data* page, was dedicated to recording demographic details of the interviewed experts. On the *TD Types and Indicators List* page, we left space under each TD Type so that experts could suggest new indicators, accompanied by brief explanations. We conducted the interviews via online platforms, each session lasting between 45 to 60 minutes. During the interviews, we shared the spreadsheet with every expert. We began each interview by clarifying the interview's purpose. Subsequently, we discussed the definitions of the technical debt types and associated indicators along with an explanation of how we compiled the TD Types and Indicators List. The experts were asked to verify if each indicator was correctly categorized under its respective TD type and, if they disagreed, to suggest either an alternative existing category or a new one. We carefully noted all feedback in a dedicated section for future revisions. A detailed overview of the experts' feedback and their contributions to the list can be found in the Results section.

Table 1: **Experts' Characteristics**

| ID | Role | Field | Years of Exp. | Project Type/Fields of Interest |
|---|---|---|---|---|
| E1 | CTO | IT & Services | 15 | Software Product Development |
| E2 | Senior Software Developer | Tourism | 12 | E-commerce Platform Development, Business Process Automation, System Development |
| E3 | Senior Software Developer | Defense | 16 | Software Product Development |
| E4 | Lead Developer | Tourism | 10 | Website Development, Mobile App Development, Cloud Migration and Integration, Software Product Development |
| E5 | Professor | Info. Systems | 30 | Computer Networks, Software Engineering, Big Data, Machine Learning, Internet of Things |

**Step 7: Revision**

The final step involved revisiting the list in light of the experts' feedback. After completing all five interviews, a consolidated spreadsheet was prepared to compare answers of the experts. Any recommended modifications, additions, or deletions were implemented to produce the final list of TD types and indicators. This finalized list serves as the key output of our research process and aims to present the most comprehensive and validated catalogue of TD types and their indicators to date.

**Threats to the Validity**

The validity threads and associated solutions are discussed below.

First, the data collection and analysis have been conducted by the authors' judgment, which could impact the consistency of mapping technical debt types and indicators. To address this threat, a clear set of criteria was established beforehand to guide the mapping process and minimize subjective bias. Second, the selection of experts for review may introduce bias, as their views are shaped by their own experiences and may not be universally applicable. To mitigate this, experts with varying backgrounds and expertise were selected to provide a wider perspective. Third, the scope of the literature review is constrained by database availability and the chosen search strings, potentially overlooking relevant studies. In order to broaden the scope, multiple databases were searched and gray literature sources were also included in the final source list. The search strings were carefully constructed to be inclusive of a wide range of relevant terms. Lastly, while the iterative process of revising the list aims to improve accuracy, there's a risk that some indicators might be overemphasized or underrepresented. To respond to this, regular cross-checks were performed during the revision process to ensure a balanced representation of indicators.

# 3 RESULTS

In this section, we present the overview of revisions and the final TD Types and Indicators List.

## 3.1 Overview of Changes

The process of refining and validating the list of Technical Debt (TD) indicators led to several modifications to the initial list. These modifications were crucial as they contribute to the enhancement of the list's comprehensiveness and accuracy. They helped to align the final list more closely with the expert opinions.

The extension of the initial list of technical debt indicators was an important aspect of our findings. Initially, we specified 60 indicators from formal literature review which was then increased to 92 indicators with inclusion of the gray literature sources. The indicators added based on gray literature sources were given as gray colored in Table 3. The remarkable expansion of the initial list represents an 91.67% increase. It is a meaningful improvement that substantially enriches the foundation for analyzing technical debt. With the total count of indicators from 60 to 101, our study has substantially bridged the identified gaps.

After contributing the initial list, the results of the expert reviews expanded the initial list further. The changes made after the expert reviews are categorized into five distinct types, each representing a specific kind of modification made to the initial list of TD indicators. The categories are given as follows: "Remained Same", "Name Changed", "Removed", "Category Changed", and "Newly Added" indicators. A significant portion of the indicators, precisely 58.78%, remained unchanged after the experts' review. This percentage shows the high level of initial accuracy in the mapping study and the authors' contribution. The approval rate of recently added TD indicators is approximately 93.75% (30 out of 32) which shows the precision and relevance of the indicators that were added by the authors. However, an observable portion was subjected to alterations for better alignment and clarity. A total of 9.16% had their names changed to better reflect the indicators' meaning. The portion of 7.63% were removed for reasons such as redundancy or irrelevance. The category of the indicators was changed for 3.05% of them to better fit their characteristics. Notably, a substantial 21.37% of the indicators were new additions to the list by the experts. This reflects the insights acquired from the expert opinions that have contributed to the final list, considerably.

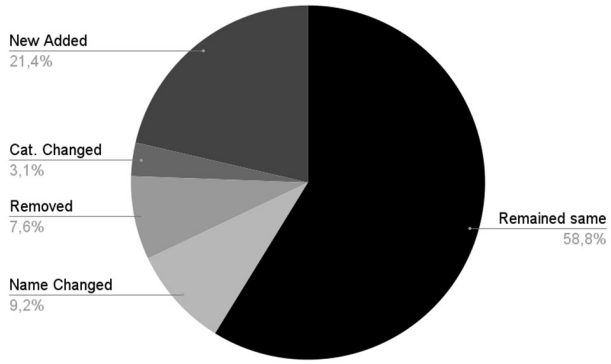


Figure 2: **Pie Chart illustrating the distribution of changes**

## 3.2 Revisions Made in Technical Debt Types and Indicators

Upon reviewing expert feedback, several adjustments were made to the initial list of 101 indicators:

***Stability of Indicators.*** Out of the 101 indicators, 77 remained unchanged.

***Category Changes.*** Two indicators shifted categories based on majority's opinion. Specifically: *(i)* "Poor release planning", originally under Infrastructure Debt, was recategorized to Management Debt. *(ii)* "Poor handling of error conditions or exceptions", initially under Code Debt, was moved to Management Debt.

***Name Adjustments.*** Twelve indicators underwent name changes due to ambiguities or incorrect phrasing. Notably, Code Debt indicators, mainly patterns or anti-patterns from Object-Oriented Programming (OOP), were rephrased for clarity. For instance, terms like "information hiding" or "encapsulation" became "lack of information hiding" or "lack of encapsulation". Contrarily, anti-patterns, such as "feature envy", remained

unchanged. Overall, eight Code Debt indicators were modified in this manner.

***Removals.*** 10 indicators were deleted due to redundancy or were replaced with clearer indicators. As an example, "inaccurate or duplicate requirements" was split into "ambiguous functional or non-functional requirements" and "incorrect functional or non-functional requirements". Also, "insufficient documentation" was removed in favor of "incomplete documentation".

***Specific Additions.*** Test Debt indicators were expanded to encompass the entire testing process, adding areas like lacking software tests. While Code Debt indicators were primarily OOP-focused, additions were made to represent other programming paradigms, introducing terms like "Spaghetti Code", "Golden Hammer" and "Boat Anchor". Requirements Debt indicators were restructured in line with the IEEE 830-19T98 Recommended Practice for Software Requirements Specifications Standard [17].

The number of changes made in each TD Type based on expert opinion is given in Table 2. In Table 2 and Table 3, we presented each TD Type with a four-character abbreviation. TD indicators were assigned two-digit number IDs linked with TD Types. The abbreviations for each TD type are:

- Architecture Debt (ARCH)
- Code Debt (CODE)
- Configuration Debt (CONF)
- Design Debt (DESG)
- Documentation Debt (DOCT)
- Infrastructure Debt (INFR)
- Management Debt (MANG)
- People Debt (PPLE)
- Requirements Debt (REQS)
- Test Debt (TEST)
-

Table 2: **Distribution of Changes by Technical Debt Types**

| TD Type CODES | Remained Same | Ind. Name Changed | Removed | Ind. Cat. Changed | Newly Added |
|---|---|---|---|---|---|
| ARCH | 8 | 1 | 1 | | 3 |
| CODE | 10 | 8 | 1 | 1 | 4 |
| CONF | 7 | | 1 | | |
| DESG | 7 | | 3 | | |
| DOCT | 3 | | 1 | | |
| INFR | 10 | | | 1 | |
| MANG | 7 | | | 1 | 6 |
| PPLE | 8 | | | | |
| REQS | 5 | 2 | 3 | | 6 |
| TEST | 12 | 1 | | 1 | 9 |
| **TOTAL** | **77** | **12** | **10** | **4** | **28** |

According to the experts' contribution, a total of 28 new indicators were added to five technical debt types which were architecture debt, code debt, management debt, requirements debt, and test debt. Test debt indicators were the most increased by 9 more indicators. It is followed by both 6 more indicators for management debt and requirements debt. The only category that remined unchanged was People Debt.

## 3.3 Final TD Indicators List

In this section, we present the finalized list of TD indicators mapped with the TD types. The list is organized by unique TD identifiers (ID), TD indicator names, their primary references in the literature, and any modifications made based on expert opinions. In the list, indicators highlighted in gray are notable contributions from the authors. These indicators are not directly sourced from formal academic literature but were instead derived from gray literature sources such as blogs, whitepapers, and industry reports.

Table 3: **Final TD Types and Indicators List**

| ID | TD Indicators | Source | Changes Made after Experts Opinion |
|---|---|---|---|
| ARCH01 | Timing or sequencing dependencies among resources | [11] | Name Changed |
| ARCH02 | violations of defined architectural styles | [5] | Remained Same |
| ARCH03 | incorrect interactions between major system components | [11] | Remained Same |
| ARCH04 | violations of principles such as separation of concerns | [5], [13] | Remained Same |
| ARCH05 | insufficient consideration for non-functional requirements | [11] | Remained Same |
| ARCH06 | violation of modularity | [3] | Remained Same |
| ARCH07 | non-uniformity of patterns and policies | [11] | Remained Same |
| ARCH08 | complex architectural behavioral dependencies | [5], [11] | Remained Same |
| ARCH09 | solutions that become sub-optimal as technologies and patterns become superseded | [4] | Remained Same |
| ARCH10 | low system operability | [18] | Remained Same |
| ARCH11 | lack of architectural analysis | Expert Opinion | Newly added |
| ARCH12 | not having architectural documentation | Expert Opinion | Newly added |
| ARCH13 | inconsistency between documented and implemented architecture | Expert Opinion | Newly added |
| CODE01 | duplicate code | [2], [5], [11] | Remained Same |
| CODE02 | incohesive code | [5] | Name Changed |
| CODE03 | inconsistent coding style that reduces the readability of code | [2] | Name Changed |

| | | | |
|---|---|---|---|
| CODE04 | poorly organized logic that makes it easy for a software solution to break when updated | [2] | Remained Same |
| CODE05 | deviations from coding standards (SOLID) | [2], [5], [13] | Remained Same |
| CODE06 | lack of information hiding | [10], [12] | Name Changed |
| CODE07 | lack of encapsulation | [10], [12] | Name Changed |
| CODE08 | improper use of inheritance | [10], [12] | Name Changed |
| CODE09 | god classes | [3], [5], [10], [12] | Remained Same |
| CODE10 | brain class/method | [12] | Remained Same |
| CODE11 | feature envy | [12] | Remained Same |
| CODE12 | undesired/high coupling | [12] | Name Changed |
| CODE13 | dispersed coupling | [12] | Remained Same |
| CODE14 | shotgun surgery | [12] | Remained Same |
| CODE15 | inconsistent naming conventions | [19] | Remained Same |
| CODE16 | unused or dead code | [19] | Remained Same |
| CODE17 | overcomplicated or convoluted algorithms or lack of algorithmic complexity | [2], [3] | Name Changed |
| CODE18 | lack of code reuse | [11] | Name Changed |
| CODE19 | lack of reusability of components | Expert Opinion | Newly added |
| CODE20 | spaghetti code | Expert Opinion | Newly added |
| CODE21 | golden hammer | Expert Opinion | Newly added |
| CODE22 | boat anchor | Expert Opinion | Newly added |
| CODE23 | poor handling of error conditions or exceptions | Gray Literature | Cat. Changed |
| CONF01 | unnecessary code forks | [3], [5] | Remained Same |
| CONF02 | multi-version support | [5] | Remained Same |
| CONF03 | inconsistent or scattered configuration files | [20] | Remained Same |
| CONF04 | lack of version control for configuration items | [20] | Remained Same |
| CONF05 | manual or ad-hoc configuration management | Gray Literature | Remained Same |
| CONF06 | overly complex or convoluted configurations | Gray Literature | Remained Same |
| CONF07 | poor visibility into configuration changes | [20] | Remained Same |
| DESG01 | inadequate or inconsistent use of data structures | Gray Literature | Remained Same |
| DESG02 | design pattern grime | [3], [5], [12] | Remained Same |
| DESG03 | design pattern rot | [12] | Remained Same |
| DESG04 | abstraction smells (imperative, multifaceted, unnecessary, unutilized, duplicate abstraction) | [14] | Remained Same |
| DESG05 | encapsulation smells (deficient, unexploited encapsulation) | [14] | Remained Same |
| DESG06 | modularization smells (broken, insufficient, cyclically-dependent, hub-like modularization) | [10], [14] | Remained Same |
| DESG07 | hierarchy smells (missing, wide, deep, rebellious, broken, multipath, cyclic, unfactored hierarchy) | [14] | Remained Same |
| DOCT01 | out-of-date documentation | [5] | Remained Same |
| DOCT02 | incomplete documentation | [3] | Remained Same |
| DOCT03 | missing documentation | [3] | Remained Same |
| INFR01 | delaying an upgrade or infrastructure fix | [3] | Remained Same |
| INFR02 | inadequate network infrastructure | Gray Literature | Remained Same |
| INFR03 | lack of disaster recovery and backup mechanisms | Gray Literature | Remained Same |
| INFR04 | insufficient scalability and capacity planning | Gray Literature | Remained Same |
| INFR05 | outdated or unsupported infrastructure components | Gray Literature | Remained Same |
| INFR06 | lack of automation and orchestration | Gray Literature | Remained Same |
| INFR07 | poor performance | Gray Literature | Remained Same |
| INFR08 | inefficient resource utilization | Gray Literature | Remained Same |
| INFR09 | inflexible or lack of adaptability infrastructure architecture | Gray Literature | Remained Same |
| INFR10 | inadequate security measures | Gray Literature | Remained Same |
| MANG01 | inadequate resource planning | Gray Literature | Remained Same |
| MANG02 | constant firefighting | Gray Literature | Remained Same |
| MANG03 | frequent budget overruns | Gray Literature | Remained Same |
| MANG04 | excess overtime | Gray Literature | Remained Same |
| MANG05 | lack of strategic alignment | Gray Literature | Remained Same |
| MANG06 | lack of skill alignment | Gray Literature | Remained Same |
| MANG07 | incorrect effort estimation | [5] | Remained Same |
| MANG08 | lack of monitoring | Expert Opinion | Newly added |
| MANG09 | lack of measurement | Expert Opinion | Newly added |
| MANG10 | lack of risk management | Expert Opinion | Newly added |
| MANG11 | lack of training | Expert Opinion | Newly added |
| MANG12 | problems with ownership management | Expert Opinion | Newly added |
| MANG13 | poor release planning | [5] | Cat. Changed |
| MANG14 | lack of task prioritization | Expert Opinion | Newly added |
| PPLE01 | expertise concentrated in too few people | [3] | Remained Same |

| | | | |
|---|---|---|---|
| PPLE02 | high employee turnover | Gray Literature | Remained Same |
| PPLE03 | low employee morale | Gray Literature | Remained Same |
| PPLE04 | ineffective communication | Gray Literature | Remained Same |
| PPLE05 | lack of collaboration and teamwork | Gray Literature | Remained Same |
| PPLE06 | inadequate leadership | Gray Literature | Remained Same |
| PPLE07 | lack of diversity and inclusion | Gray Literature | Remained Same |
| PPLE08 | absence of knowledge sharing | Gray Literature | Remained Same |
| REQS01 | partially implemented functional requirements | [3] | Name Changed |
| REQS02 | lack of non-functional requirements specification | [3] | Name Changed |
| REQS03 | lack of traceability between requirements and implementation | [3] | Remained Same |
| REQS04 | changing requirements without proper impact analysis | Gray Literature | Remained Same |
| REQS05 | lack of user involvement in requirements elicitation | Gray Literature | Remained Same |
| REQS06 | lack of requirements review or validation | Gray Literature | Remained Same |
| REQS07 | scope creep | Gray Literature | Remained Same |
| REQS08 | incorrect functional or non-functional requirements | Expert Opinion | Newly added |
| REQS09 | ambiguous functional or non-functional requirements | Expert Opinion | Newly added |
| REQS10 | incomplete functional or non-functional requirements | Expert Opinion | Newly added |
| REQS11 | inconsistent functional or non-functional requirements | Expert Opinion | Newly added |
| REQS12 | unverifiable functional or non-functional requirements | Expert Opinion | Newly added |
| REQS13 | inflexible functional or non-functional requirements | Expert Opinion | Newly added |
| TEST01 | planned tests that were not run/lack of functional testing | [3] | Remained Same |
| TEST02 | low code coverage | [2], [3], [5] | Remained Same |
| TEST03 | lack of test automation | [2], [3] | Remained Same |
| TEST04 | residual defects not found in tests | [3], [5] | Remained Same |
| TEST05 | expensive tests | [5] | Remained Same |
| TEST06 | lack of test environment | Gray Literature | Name Changed |
| TEST07 | lack of/poor test data management | Gray Literature | Remained Same |
| TEST08 | Inadequate performance testing | [21] | Remained Same |
| TEST09 | lack of regression testing | [21] | Remained Same |
| TEST10 | poor test case design | [3] | Remained Same |
| TEST11 | limited usability testing | [21] | Remained Same |
| TEST12 | inadequate security testing | [21] | Remained Same |
| TEST13 | outdated test cases | [3] | Remained Same |
| TEST14 | lack of unit/module/component tests | Expert Opinion | Newly added |
| TEST15 | lack of integration tests | Expert Opinion | Newly added |
| TEST16 | lack of acceptance tests | Expert Opinion | Newly added |
| TEST17 | lack of system tests | Expert Opinion | Newly added |
| TEST18 | lack of maintenance testing | Expert Opinion | Newly added |
| TEST19 | inadequate portability testing | Expert Opinion | Newly added |
| TEST20 | lack of static testing | Expert Opinion | Newly added |
| TEST21 | lack of chaos engineering | Expert Opinion | Newly added |
| TEST22 | lack of conducting drills for disaster recovery | Expert Opinion | Newly added |

# 4 DISCUSSION

In this study, a comprehensive technical debt (TD) types and their indicators was explored. The procedure of mapping and expert opinion led to an enriched TD list with 120 TD indicators associated with 10 TD Types. The initial list was enhanced by 21.37%. TD categorization was extended beyond common categories to include newly introduced categories, "Management Debt" and "Configuration Debt". These additions demonstrate the potential for a more holistic approach in identifying and managing technical debt in software projects. The employed methodology emphasizes the importance of combining empirical data with industry expertise to achieve a comprehensive understanding. Despite the methodical approach, the study acknowledges certain validity threats before mentioned. The discussion with experts validated the compiled list. Also, it brought invaluable insights into the practical implications and challenges faced in managing technical debt. This study, not only contributes to the academic understanding of technical debt but also develops informative approaches in tackling technical debt in software projects.

# 5 CONCLUSIONS

There is no doubt that technical debt is a potential threat for software development organizations if it is not paid. For effective repayment of TD, organizations need to identify what TD types they do have. This research underlines the critical necessity of addressing technical debt by indicating causes of TD. We aimed to fill a significant gap in existing literature by providing a comprehensive list of technical debt indicators. Consequently, we now present 10 distinct TD categories, associated with a comprehensive set of 120 TD indicators, designed to assist in the identification, resolution, and management of Technical Debt. Our methodology was adopted to combine mapping with expert consultations to produce the final list. We offer a valuable resource for software engineers, researchers, and IT managers aiming to more effectively identify, manage, and ultimately mitigate technical debt.

As future research direction, organizations can apply our extended list of TD indicators in real-world software projects across various domains. The assessment of TD indicators realization and their impact on project outcomes can be assessed through case studies. These empirical findings will enhance the list's practical value for both industry professionals and researchers.

## ACKNOWLEDGMENTS

We would like to thank the experts who generously contributed their time and insights during the interview phase of this research. Their expertise greatly enriched the quality and depth of our findings. We also thank our colleagues and reviewers who provided invaluable feedback and guidance throughout the course of this study.

## REFERENCES

[1] Ward Cunningham. 1992. The WyCash Portfolio Management System. In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) (OOPSLA '92). Association for Computing Machinery, New York, NY, USA, 29–30. DOI: https://doi.org/10.1145/157709.157715

[2] Edith Tom, Aybüke Aurum, Richard Vidgen. 2013. An exploration of technical debt. Journal of Systems and Software 86, 6 (2013), DOI: https://doi.org/10.1016/j.jss.2012.12.052

[3] Nicolli S.R. Alves, Thiago S. Mendes, Manoel G. de Mendonça, Rodrigo O. Spínola, Forrest Shull, Carolyn Seaman. 2016. Identification and management of technical debt: A systematic mapping study. Information and Software Technology. 100-121. DOI: https://doi.org/10.1016/j.infsof.2015.10.008

[4] Nicolli S. R. Alves, Leilane F. Ribeiro, Vivyane Caires, Thiago S. Mendes, and Rodrigo O. Spínola. 2014. Towards an Ontology of Terms on Technical Debt. In Proceedings of the 2014 Sixth International Workshop on Managing Technical Debt (MTD '14). IEEE Computer Society, USA, 1–7. https://doi.org/10.1109/MTD.2014.9

[5] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A systematic mapping study on technical debt and its management. Journal of Systems and Software. 101, C (March 2015), 193–220. DOI: https://doi.org/10.1016/j.jss.2014.12.027

[6] C.B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering, 25, 4, 557-572. DOI: 10.1109/32.799955.

[7] Kai Petersen, Robert Feldt and Shahid Mujtaba, Michael Mattsson. 2008. Systematic Mapping Studies in Software Engineering. 12th International Conference on Evaluation and Assessment in Software Engineering (EASE), DOI: 10.14236/ewic/EASE2008.8

[8] Roger M. Cooke. 1992. Experts in uncertainty: Opinion and subjective probability in science. Oxford University Press.

[9] Valentina Lenarduzzi, Terese Besker, Davide Taibi, Antonio Martini, Francesca Arcelli Fontana. 2021. A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools. Journal of Systems and Software, 171, 110827. DOI: https://doi.org/10.1016/j.jss.2020.110827.

[10] Clemente Izurieta, Antonio Vetro', Nico Zazworka, Yuanfang Cai, Carolyn Seaman, Forrest Shull. 2012. Organizing the technical debt landscape. In Proceedings of the Third International Workshop on Managing Technical Debt (MTD '12). IEEE Press, 23–26. DOI: 10.1109/MTD.2012.6225995

[11] Antonio Martini, Jan Bosch, Michel Chaudron. 2015. Investigating Architectural Technical Debt accumulation and refactoring over time: A multiple-case study. Information and Software Technology, 67, 237-253. DOI: https://doi.org/10.1016/j.infsof.2015.07.005.

[12] Nico Zazworka, Antonio Vetro', Clemente Izurieta, Sunny Wong, Yuanfang Cai, Carolyn Seaman, Forrest Shull. 2013. Comparing Four Approaches for Technical Debt Identification. Software Quality Journal. 22. 1-24. DOI: 10.1007/s11219-013-9200-8

[13] Woubshet Nema Behutiye, Pilar Rodríguez, Markku Oivo, Ayşe Tosun. 2017. Analyzing the concept of technical debt in the context of agile software development: A systematic literature review. Information and Software Technology, 82, 139-158. DOI: https://doi.org/10.1016/j.infsof.2016.10.004

[14] Tushar Sharma, Paramvir Singh, Diomidis Spinellis. 2020. An empirical investigation on the relationship between design and architecture smells. Empirical Software Engineering, 25, 5, 4020–4068. DOI: https://doi.org/10.1007/s10664-020-09847-2

[15] Francesca Arcelli Fontana, Valentina Lenarduzzi, Riccardo Roveda, Davide Taibi. 2019. Are architectural smells independent from code smells? An empirical study,
Journal of Systems and Software, 154, 139-156. DOI: https://doi.org/10.1016/j.jss.2019.04.066.

[16] Nico Zazworka, Michele A. Shaw, Forrest Shull, and Carolyn Seaman. 2011. Investigating the impact of design debt on software quality. In Proceedings of the 2nd Workshop on Managing Technical Debt (MTD '11). Association for Computing Machinery, New York, NY, USA, 17–23. https://doi.org/10.1145/1985362.1985366

[17] IEEE 830-19T98 Recommended Practice for Software Requirements Specifications Standards

[18] J. Garcia, D. Popescu, G. Edwards and N. Medvidovic, "Toward a Catalogue of Architectural Bad Smell", Proceedings of the 5th International Conference on the Quality of Software Architectures: Architectures for Adaptive Software Systems (QoSA), pp. 146-162, 2009. DOI:10.1007/978-3-642-02351-4_10

[19] X. Han, A. Tahir, P. Liang, S. Counsell and Y. Luo, "Understanding Code Smell Detection via Code Review: A Study of the OpenStack Community," 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), Madrid, Spain, 2021, pp. 323-334, doi: 10.1109/ICPC52881.2021.00038.

[20] L. A. Rosser and J. H. Norton, "A Systems Perspective on Technical Debt," 2021 IEEE Aerospace Conference (50100), Big Sky, MT, USA, 2021, pp. 1-10, doi: 10.1109/AERO50100.2021.9438359.

[21] Guo, Y., Spínola, R.O. & Seaman, C. Exploring the costs of technical debt management – a case study. Empir Software Eng 21, 159–182 (2016). https://doi.org/10.1007/s10664-014-9351-7