Received: 11 July 2022 | Revised: 31 August 2022 | Accepted: 28 September 2022
DOI: 10.1002/smr.2516

REVIEW ARTICLE

Journal of Software: Evolution and Process WILEY

The Pandora's box of social, process, and people debts in software engineering

Muhammad Ovais Ahmad[1] | Tomas Gustavsson[2]

[1]Department of Computer Science, Karlstad University, Karlstad, Sweden
[2]Karlstad Business School, Karlstad University, Karlstad, Sweden

Correspondence
Muhammad Ovais Ahmad, Department of Computer Science, Karlstad University, Karlstad, Sweden.
Email: ovais.ahmad@kau.se

[Correction added on 18 October 2022, after first online publication: ORCID of second author has been added in this version.]

Abstract

In software engineering, technical debt (TD) has been widely investigated, but debt regarding social issues, people, and processes has not been explored as much. It should be noted here that we use nontechnical debt (NTD) as an umbrella term to cover *social, process, and people debts*. Although the number of studies on NTD in software is increasing, the majority of them are descriptive rather than rigorous, and there is no systematic development of cumulative knowledge. As a result, identifying the fundamental causes of NTD and the associated mitigation techniques in software engineering is challenging. Therefore, this study investigates the scientific evidence regarding NTD till date by conducting a systematic mapping review of software engineering research between January 2000 and October 2021. The search strategy resulted in 175 studies, 17 of which were identified as unique and relevant primary papers. The primary studies show that NTD and TD are inextricably linked. In addition, this study also captured a plethora of causes and mitigation strategies for managing NTD and thus makes four important contributions: (i) highlighting state-of-the-art NTD research; (ii) identification of the reported causes and mitigation strategies in the primary papers; and (iii) determination of opportunities for future NTD research.

KEYWORDS
agile, people debt, process debt, social debt, software development, systematic mapping review

1 | INTRODUCTION

Software companies have always striven to become more responsive in terms of identifying and satisfying customers' needs. One strategy for increasing responsiveness is the introduction of autonomous teams and agile software development.[1] Lean and agile methods have had a lot of success.[2] Specifically, these methods focus on swiftly responding to change, constant deliveries, individual interactions, customer collaboration, and so on. However, such a swiftly value-driven demanding environment is a breeding ground for suboptimal processes that might have short-term benefits but an overall negative impact on the organization in the medium-long term. Examples of negative impacts can be duplicated work, misunderstandings, and integration problems. In this context, a financial metaphor can be used to successfully describe such phenomena: Aiming for short-term goals is equivalent to getting into ***debt***, and the additional negative impact in the long term is considered as the interest that is paid. Whereas the term "debt" originated in the accounting domain and refers to a sum of money that is owed or due,[3] in software engineering, "technical debt" is used as a metaphor.

In 1992, Cunningham introduced technical debt (TD) to the software community as “the debt incurred through the speeding up of software project development which results in a number of deficiencies ending up in high maintenance overheads.”[3] Avgeriou et al[4] further elaborated on this definition: “In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability.”[4]

In software engineering, this metaphor is used to describe prevalent issues including TD (i.e., code debt and smells) and other debt (i.e., process, social, and people debts).[5,6] Moreover, TD is no longer restricted to issues with source code and is now used to describe situations where developers make sacrifices in the development or take shortcuts in order to optimize other things, such as implementing necessary features before a deadline,[7] for example, “sneaking around an abstraction because it is too hard (or impossible) to figure how to ‘do it right’, skipping or scrimping on documentation (both in the code and external documentation).”[8] Further, its application has evolved and extended to other artefacts (e.g., design, testing, and documentation) of the entire software development life cycle.[7] Thus, TD is a critical issue in the software industry. Below, we have provided real-life examples of the significant costs incurred by TD. In 2010, Gartner estimated the global TD bill to be $US500 billion, with the potential to double in 5 years.[6] The CISQ 2020 report determined that the total cost of poor software quality in the US was $2.08 trillion, and TD resulting in the need for the correction of severe defect would need another $1.31 trillion (minus interest).[9] Meanwhile, in the case of the Danish government, a third of critical IT systems are in inadequate condition, for example, out-of-date or with non-existing documentation, lack IT security, and outdated software and hardware needs upgradation.[10] Regarding Sweden's government agencies, 70% of their IT systems were outdated.[11]

However, TD is only one side of the coin because software development is a sociotechnical phenomenon. Specifically, it is a structural barrier,[12] and management is responsible for decisions that cause it.[13] “In the sense, technical debt is incurred in one discipline, but it is burdened and has to be repaid in another discipline”[13] (p. 40). Further, according to Klinger et al,[14] “the decision to acquire technical debt is not made by technical architects, but rather by non-technical stakeholders who cause the project to acquire new technical debt or discover existing technical debt that wasn't previously visible” (p. 35). In this context, Kruchten et al[15] explained that some aspects of TD are visible (i.e., new functionality to be added), whereas others are invisible (natural software ageing and evolution). Further, they elaborated that the major causes of TD are schedule pressure, carelessness, lack of education, poor processes, non-systematic verification of quality, or basic incompetence.[16,17]

Moreover, software projects' success or failure is a combination of both technical and nontechnical aspects.[18,19] However, both researchers and practitioners paid a lot of attention to TD in comparison with other types of debt in software projects. Nevertheless, software development is a human-centered sociotechnical activity that has a symbiotic relationship with the environment and the way it operates as well as interlinked people or stakeholders. In this context, Appendix A offers a glossary of the 15 types of debt and their definitions. Some of them are purely technical, such as Defect TD, whereas others depend on a combination of social and technical aspects, such as requirements and design debts. The final three types of debt listed in Appendix A (*process, social, and people debts*) mainly cover nontechnical or social aspects of software development. In this paper, we use “nontechnical debt” (NTD) as an umbrella term to cover these three types.

In light of the existing literature, it is evident that during the last 10 years, TD has been heavily investigated to understand its functioning from various perspectives. For example, TD effort,[6] tools,[20] and types, management strategies[21] and managing architectural TD[22] as well as TD in terms of agile development,[23] management elements,[24] and prioritization,[5] etc. Meanwhile, NTD has remained latent and relatively unexplored. To better understand debt, there is a need for investigation to consolidate NTD literature in software engineering. To the best of our knowledge, no systematic mapping review has been conducted on NTD, which covers *process, social, and people debts*.

The overarching goal of this study is to identify and structure the body of knowledge of NTD in software engineering by conducting a systematic mapping review. The aims of this study are as follows:

- Examine the latest research on selected NTD in software engineering
- Identify and analyze the causes of NTD, along with mitigation strategies
- Identify the opportunities for future research on NTD

The rest of the paper is organized as follows. In Section 2, the process (e.g., planning, conducting, and reporting) of the systematic mapping review is presented. Then, in Section 3, we conduct a search for the latest and most relevant NTD research. Section 4 offers details of reported NTD definitions and causes with examples. Section 5 compiles a plethora of NTD mitigation strategies. Sections 6 and 7 provide a discussion of our results and future research direction, respectively. Finally, the validity threats and limitations of the study are discussed in Sections 8 and 9 to conclude the paper.

2 | RESEARCH METHOD

This section presents the systematic mapping process. In this study, we follow the guidelines provided by Kitchenham et al[25] and Petersen et al,[26] which have three main phases: planning, conducting, and documenting. Each phase of the mapping study is discussed in detail in the rest of the section.

2.1 | Planning the mapping study

The impetus for conducting a systematic mapping study is to focus on the “classification and thematic analysis of literature on a software engineering topic”[11] (p. 640). In this context, Kitchenham et al[25] provided a detailed comparison between a systematic literature review and a systematic mapping study. A typical systematic literature review is motivated by a specific research topic that may be answered empirically. On the other hand, a mapping study examines a wider range of software engineering issues and categorizes the key research publications in a certain area.[25] The research topics in mapping studies are relatively intricate, such as those in which empirical methodologies were employed. Specifically, the motivation for conducting this mapping study is to identify, classify, and analyze the current state of NTD in the context of software engineering. To achieve this broad research objective, the research questions detailed in Table 1 will be answered.

As RQ1 is a broad research question, five sub-questions (RQ1.1–RQ1.5) have been identified as being pertinent to answering this question. RQ2 and RQ3 will then provide a synthesis of the reported NTD causes as well as management strategies in the software engineering domain. Next, RQ4 will identify potential research gaps in the context of NTD to guide future research initiatives.

2.2 | Conducting the mapping study

At first, we conducted a pilot search on debt in software engineering using Google Scholar. The aim was to determine the existence of any secondary studies on the given topic. We conducted this search using the following string: (“process debt” OR “social debt” OR “people debt” OR “technical debt”) AND (“systematic mapping” OR “mapping study” OR “systematic literature review”). The search was conducted in October 2021, and Google Scholar yielded 663 results. It was evident from the search that the current focus of research is predominantly on TD, whereas not a single review was found on NTD. Therefore, we conducted a systematic mapping review to explore and provide insight into NTD using the guidelines defined by Kitchenham et al[25] and Petersen et al.[26]

The search string was formulated based on the population, intervention, comparison, and outcome (PICO) criteria suggested by Kitchenham and Charters.[27] Nevertheless, we relied on the population AND intervention combination while omitting the comparison and outcomes facets of PICO because our aim is to cover NTD, which is a broader research topic and avoid limiting the outcomes or comparing the interventions. Specifically, population refers to the application area that is *software*, and intervention refers to NTD (*process*, *people, and social debts*). The former is the expected search that will include all documents with the word “software” in the title, abstract, or keywords. It was executed in six databases: *ACM Digital Library*, *IEEE Xplore Digital Library*, *ScienceDirect*, *Scopus*, *SpringerLink*, and *Web of Science*. The selected databases are recognized and pertinent to this study, as these return the most publications in the software engineering domain.[27] The selected databases and retrieved papers are listed in Table 2.

In total, the search string used across six databases retrieved 175 publications. Two authors independently analyzed the 175 publications to remove the following: (i) duplicate papers; (ii) non-English publications; (iii) non-software engineering studies; and (iv) non-peer-reviewed scientific papers. Next, the authors conducted in-depth reviews of each paper; this required the researchers to read the titles, abstracts, introductions, and conclusions. The outcome of this process was 17 primary studies, which were then quality assessed using the 11-factor criteria proposed by Dybå and Dingsøyr.[28] The primary studies (S) are listed in Appendix B and are identified by the symbol (S).

TABLE 1 Research questions of this study

Research question	Description
RQ1. What is the current state of the research on the selected NTD type in software engineering? RQ1.2. What is the number of academic studies on NTD that have been published between 2000 and 2021? RQ1.3. What publication channels have been used to publish these studies? RQ1.4. What research methods have been used in the published studies? RQ1.5. What is the quality of the published papers?	Understanding the rhetoric in contemporary NTD research is the key research objective of this mapping study. With this, we will be able to map the present study topics and obtain a comprehensive grasp of NTD research through the compilation of all pertinent publications from scientific sources. This mapping of various types of NTD will enable other academics and practitioners to better understand the current research foci, which will serve to advance future research
RQ2. What are the reported causes of NTD? RQ3. What are the reported NTD prevention strategies?	RQ2 and RQ3 will provide a synthesis of the reported causes of NTD and mitigation strategies in the software engineering domain
RQ4. What are the future research directions for NTD in software engineering?	RQ4 will provide a detailed future research agenda on the NTD topics in software engineering

TABLE 2 Selected databases and retrieved papers

Database	Hits	Filters
ACM	46	None
SpringerLink	44	Computer Science only; English
Web of Science	9	None
ScienceDirect	22	Computer Science
IEEE Xplore	16	None
Scopus	38	English
Total	175	

TABLE 3 Inclusion and exclusion criteria

Inclusion criteria	Exclusion criteria
• Publications that contribute to increasing the knowledge in the NTD area • Peer-reviewed publications • Scientific papers are written in English • Studies published between January 1, 2000, and November 1, 2021 • Studies directly answering one or more of the research questions of this study • If the study was published in more than one journal or conference, the most recent version of the study was included	• Studies not related to NTD • Duplicates of the same study (in this case, the most complete version of the study was selected) • Studies not related to NTD • Studies published in non-scientific platforms such as prefaces, article summaries, overhead presentations, interviews, short papers, simulation studies, tutorials, theses, books, and book chapters

2.2.1 | Inclusion and exclusion criteria

The inclusion and exclusion criteria in Table 3 were used to ensure the inclusion of relevant studies that are within the scope of this study and that address the research questions. In addition, the studies that presented empirical data were eligible for inclusion in this mapping review when there was clear evidence of research rigor.

2.2.2 | Identification of primary studies

Figure 1 illustrates the paper selection process used in this study. This process adopted the best practices proposed by Dybå and Dingsøyr[28] and Kitchenham et al.[25] First, we removed duplicates from the entire list of studies obtained from the aforementioned six databases. In total, 175 studies were obtained, of which 39 were duplicates. Each author then independently applied the inclusion and exclusion criteria. Two authors then discussed disagreements and jointly agreed on inclusion or exclusion decision.

In the second step, we applied the inclusion and exclusion criteria to the remaining 136 studies. Two authors read the titles, abstracts, and keywords of the remaining 136 studies and examined their relevance to this mapping review. In the case of an unclear paper, it was moved to the complete reading list for the next step. At the end of this activity, 76 studies out of 136 were included in the next step, which is a complete reading and on which the analysis is based. At the end of this activity, 57 papers were excluded, and 19 were selected for quality assessment. Subsequently, two researchers independently applied the quality assessment criteria. Two papers were excluded because they did not fulfil the assessment criteria. Finally, at the end of this step, we obtained 17 primary studies for this systematic mapping review.

2.2.3 | Data extraction and analysis

After careful selection, all primary studies were subjected to an in-depth analysis. We used the technique of researcher triangulation and the formulation of specific data extraction definitions to avoid researcher bias. All primary studies were evaluated for quality (e.g., research rigor and relevance) as well as study characteristics (e.g., study type, method, domain, pertinence, publication channel, etc.). Each primary study was individually examined by two researchers, who then integrated their peer-reviewed findings into the analysis and reporting of workshop settings.

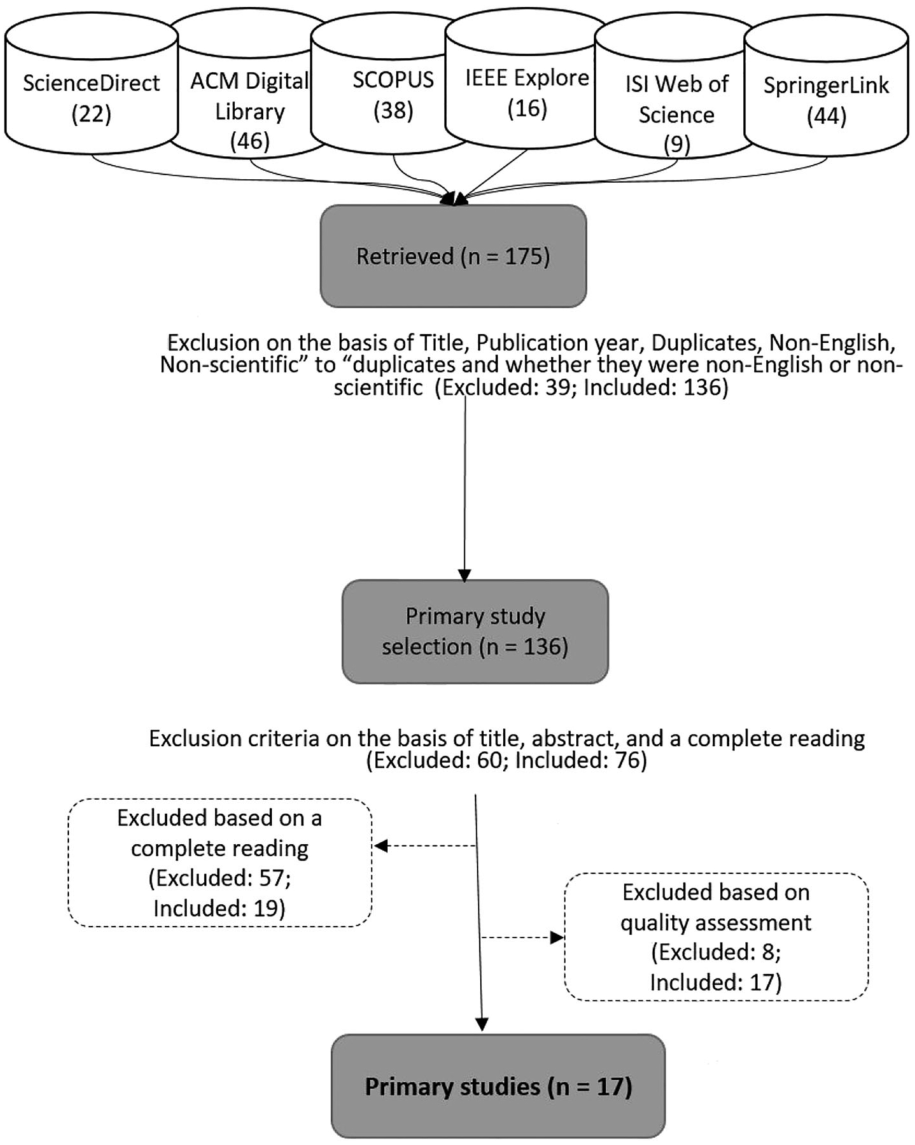
ScienceDirect (22)
ACM Digital Library (46)
SCOPUS (38)
IEEE Explore (16)
ISI Web of Science (9)
SpringerLink (44)
Retrieved (n = 175)
Exclusion on the basis of Title, Publication year, Duplicates, Non-English, Non-scientific" to "duplicates and whether they were non-English or non-scientific (Excluded: 39; Included: 136)
Primary study selection (n = 136)
Exclusion criteria on the basis of title, abstract, and a complete reading (Excluded: 60; Included: 76)
Excluded based on a complete reading (Excluded: 57; Included: 19)
Excluded based on quality assessment (Excluded: 8; Included: 17)
Primary studies (n = 17)

FIGURE 1 The paper selection process

This method assisted us in discussing and resolving any disagreements. Finally, one author had a panoptic vision of the entire mapping study as well as each activity. It helped in ensuring the consistency of the analysis, consolidating results, and reporting.

2.2.4 | Quality assessment

To assess the quality of primary studies, we applied the 11-factor quality assessment criteria (see Table 4) proposed by Dybå and Dingsøyr.[28] Each of the criteria questions was graded on a 5-point Likert scale (where 0 = *poor* and 4 = *excellent*). This scaling strategy was adopted from Lenarduzzi et al.[5] Next, two researchers independently assessed the primary studies to limit the degree of subjectivity. A paper needed to achieve a rating of higher than or equal to "2" to be considered a primary study. The result is a more objective quality assessment of all primary studies, which is presented in the analysis section of the paper.

3 | RESULTS

The results and findings of the primary studies' analysis are presented according to the previously mentioned research questions (Section 2, Table 2). The findings in this section represent the current state of NTD research in software engineering. Specifically, Section 3 provides the

TABLE 4 Quality assessment questions

No.	Quality question
1.	Is this a research paper? (or is it merely a "lessons learned" report based on expert opinion)
2.	Is there are a clear statement of the aims of the research?
3.	Is there an adequate description of the context in which the research was carried out?
4.	Was the research design appropriate to address the aims of the research?
5.	Was the recruitment strategy appropriate to the aims of the research?
6.	Was there a control group with which to compare treatments?
7.	Was the data collected in a way that addressed the research issue?
8.	Was the data analysis sufficiently rigorous?
9.	Has the relationship between researcher and participants been considered to an adequate degree?
10.	Is there a clear statement of findings?
11.	Is the study of value for research or practice?

Source: Dybå and Dingsøyr.[28]

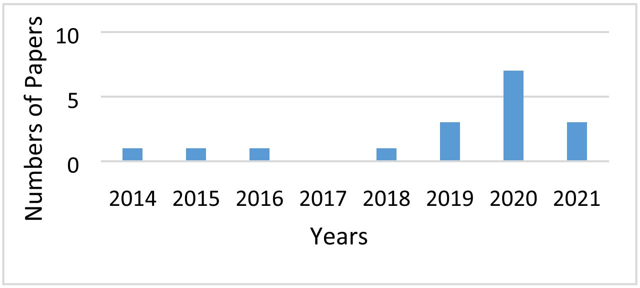

FIGURE 2 Publication by year

following: (i) publication by year; (ii) publication channel; (iii) research method used; and (v) reporting quality assessment of primary papers. Subsequently, Section 4 provides NTD types, their definitions, and causes. In Section 5, each NTD type and its prevention strategy is presented, after which in Section 6, future research directions are listed.

3.1 | Publication by year

By analyzing NTD literature published between 2000 and 2021, we identified 17 primary studies (S#) based on our adopted selection process. Figure 2 shows the number of primary studies per year. The publication trend provides a valuable insight into the fact that academic research on the topic is gaining momentum and the number of articles has been increasing in recent years. Based on our inclusion and exclusion criteria, we were not able to find empirical studies between 2000 and 2013. It is also evident that the majority of NTD studies have been undertaken in recent years (between 2019 and 2021). This is an indication that practitioners and academia are taking an interest in such topics due to the socio-technical nature of software and systems development.

3.2 | Publication channel

The selected papers on NTD have been published in various peer-reviewed journals and international conferences. Table 5 shows the main peer-reviewed publication channels for NTD research. In total, seven papers were published in journals, and 10 in international conferences.

3.3 | Research methods adopted in primary studies

We categorized the available literature on NTD according to the adopted research approaches. Table 6 offers a diverse picture of the research method used in our primary studies. Specifically, our mapping study focused only on empirical studies on NTD. A majority of the primary studies

TABLE 5 Primary studies by target journal and conference

Channel	Title	No. of publications	Primary study
Journal ($n = 7$)	*IEEE Software*	1	S12
	Journal of Systems and Software	2	S5, S7
	Empirical Software Engineering	1	S14
	Requirements Engineering	1	S13
	Journal of Internet Services and Applications	1	S17
	IEEE Transactions on Computational Social Systems	1	S6
Conference ($n = 10$)	Annual ACM Symposium on Applied Computing	1	S1
	Brazilian Symposium on Software Engineering	1	S2
	International Conference on Software Engineering: Software Engineering in Society	1	S9
	International Conference on Technical Debt	1	S3
	Asia-Pacific Software Engineering Conference	1	S4
	Hawaii International Conference on System Sciences	2	S8, S10
	International Conference on Agile Software Development	1	S11
	European Conference on Software Architecture	1	S15
	International Conference on Product-Focused Software Process Improvement	1	S16

TABLE 6 Research methods adopted in primary studies

Research method	Technique	Primary study
Quantitative	Survey, descriptive statistics	S1, S3, S5, S14
Qualitative	Semi-structured interviews	S2, S4, S9, S16
Mixed	Survey, semi-structured interviews	S7, S10, S13
	Semi-structured interviews, project documentation	S8
	Interviews, focus groups, and workshops	S12, S17
	Interviews, observations, and document analysis	S11
	Survey, workshops	S15
	Surveys, controlled experiment	S6

adopted a mixed-method approach where data collection included surveys, semi-structured interviews, focus groups, workshops, observations, and document analyses. Meanwhile, only four studies adopted a qualitative research approach and conducted semi-structured interviews in single or multiple case studies.

3.4 | Quality assessment of primary papers

Here, we assess the quality of our primary studies using the 11-factor framework proposed by Dybå and Dingsøyr.[19] Two researchers independently assessed the 17 primary studies and discussed the disagreements in a workshop. The findings of our quality assessment discussion and comparison are aggregated in Table 7.

All of the selected primary studies describe their research aim and largely provide the necessary context of their studies. Among these, there was only one that ran an experiment, and it provided a detailed description of the study participants and compared the treatments where applicable. Additionally, all the studies also provided detailed information about "data collection." However, although the "data analysis" criteria were not fully reported in every primary study, we considered this to be adequate reporting. Further, the values and findings were reported in a clear manner in primary studies. On the other hand, the relationship between the investigators and participants is not clear in most cases. More specifically, four studies did not cover this issue anywhere in their respective articles. Overall, none of the primary studies received full points based on our quality assessment framework. Nevertheless, it should be noted that a majority of the primary studies are of high quality and are published in reputable scientific peer-reviewed publications.

TABLE 7 Quality assessment of primary papers

S#	Research	Aim	Context	Design	Sampling	Control	Data collection	Data analysis	Reflexivity	Findings	Value
S1	3	4	4	4	4	0	4	2	3	4	4
S2	3	4	4	4	2	0	4	3	3	4	4
S3	3	4	4	4	2	0	4	2	4	4	4
S4	3	4	2	2	4	0	4	2	2	4	3
S5	4	4	4	4	4	0	4	4	4	4	4
S6	4	4	4	4	4	0	4	4	4	4	4
S7	4	4	2	4	4	0	4	4	4	4	4
S8	3	4	2	4	4	0	4	1	1	4	2
S9	3	4	4	2	2	0	2	2	2	2	2
S10	3	4	4	4	4	0	4	4	4	4	4
S11	3	4	4	4	4	0	4	2	2	2	2
S12	4	4	4	2	4	0	2	2	1	4	4
S13	4	4	4	4	4	0	4	4	4	4	4
S14	4	4	4	4	4	0	4	4	1	4	2
S15	3	4	4	2	2	0	2	2	1	4	3
S16	3	4	4	2	2	0	4	2	1	4	3
S17	4	4	4	4	4	0	4	4	4	4	4

4 | NTD TYPES, THEIR DEFINITIONS, AND CAUSES

This section presents the definitions of the three NTDs (i.e., process, people, and social debts) along with detailed descriptions of their causes in terms of software development.

4.1 | Process debt

Process debt is “a sub-optimal activity or process that might have short-term benefits but generates a negative impact in the medium-long term.”[S11] It also “refers to inefficient processes, e.g., what the process was designed to handle may be no longer appropriate.”[29]

First, we will technically define process debt in the context of this paper. A process is a series of repetitive operations that must be completed to achieve specified organizational goals. For example, it is one that is used to implement a testing strategy. In this context, process debt can be referred to as inefficient processes and include situations in which what the process was designed to handle may be no longer appropriate.[29] Similar to TD, process debt can provide short-term benefits but is generally detrimental to a software business in the longer run. An example here could be a team conducting daily stand-up meetings with a focus on updating the leader, where the short-term benefit is that the team satisfies leaders, but in the long run, the team circles around reporting and is not focused on sharing knowledge and solving dependencies.[S11]

Martini et al[S4] investigated process debt for the first time and reported a number of its causes, a few mitigation techniques, and its subtypes as listed here: role related—mismatching roles and responsibilities; process related—documentation, synchronization, activity-specific, and infrastructure debts as well as process unsuitability. However, the results need to be taken with caution because existing literature on software process improvement and change management challenges has many similarities with the challenges of process debts. Overall, in this mapping study, the goal is to explore and map the existing literature on process debt in terms of understanding the phenomenon and its causes and effects. In this type of debt, there are two main actors (i.e., process designers and process executors) and four key accumulation patterns (i.e., suboptimal process design, process divergence, tools, and infrastructure deficiencies).[S4] Table 8 summarizes process debt challenges and their causes.

4.1.1 | Organizational challenges

The *organization* itself is predominantly a major contributor to process debt, which has both short-term and long-term negative effects. In this context, Martini et al[S4] highlighted that *process sub-optimality can actually often lead to waste (of time, effort, budget, etc.) for stakeholders*. For example, a company may change a process based on their financial estimates that later turn out to be incorrect and thus cause *fear* and extreme

TABLE 8 Process debt causes

Process debt theme	Causes	S3	S4	S5	S10	S11	S12	S13	S16
Organizational challenges	Process competencies		X	X					
	Lack of engagement in development						X		
	Lack of follow-op assessment		X	X					
	Business decisions								X
	Lack of software culture		X						
	Suboptimal process design								
	A mismatch between requirements on different abstraction levels							X	
	Lack of time and tight schedules for delivery	X							X
	Cost of process changes		X						
	Organizational causes		X						
	Focus on producing more at the expense of quality			X					
	Excessive conformity to standards						X		
	Lack of innovation						X		
	Lack of coordination					X			
Process divergence	Informality excess				X		X		
	Value neglection		X						
	Lack of curiosity						X		
	Process divergence		X						
	Lack of coding standards and guides			X					X
	Using a formal structure to achieve community goals				X				
	Lack of process quality	X		X					
	Non-adoption of good practices			X					
	Not effective project management			X					
	Inappropriate planning			X					
	Rules for adequate configuration management (versioning debt)	X							
	Lack of prioritization		X	X					
External dependencies	(Expectation) Increased customer satisfaction								X
	Lack of knowledge about future changes								X
	Special contexts and situations		X						
	Technology and tools		X	X					
	External trends		X						

caution while handling process debt.[S4] Furthermore, introducing new conflicting processes without careful planning generates a substantial overhead. This indicates that a *lack of process competencies* (which include design, management, and execution processes) is another source of debt. In addition, a *lack of software culture* is a major cause of process debt in an organization dealing with mechanical and electrical engineering as well as the development of other cyber–physical systems. There are two main reasons: First, a process design made for the mechanical and electrical engineering context might not be optimal for software development; second, *individuals have different levels of experience, maturity, and motivation as well as leadership styles*.[S4] Regarding the second reason, one of the inferences of such challenges can be linked to a *lack of coordination and engagement in the development* of people in the organization. The experienced people might ignore or optimize some activities without following given guidelines, whereas those who are inexperienced follow a process too rigidly or lean toward *excessive conformity to standards* as they are confused. Thus, inter-team coordination is necessary when documentation and tools are used across teams.[S11] According to literatures,[S3, S5] the quality of the product suffers and process regresses when an organization focuses on *producing more* and *ensuring that a product is put on the market faster*. These aforementioned challenges are directly linked to the *lack of a follow-up process assessment*. Specifically, the negative effect here is that organizations are unable to identify and track possible challenges, and process divergence leads to the accumulation of process debt. Finally, *business decisions* generate pressure on the development team and affect the amount of debt, as explained in literature[S16]: "Previous years' business decisions have affected us a lot. The reason for this that we have a release every two months and when we have a bigger release, we just do not have enough time to do it. This means that we are in a hurry at the end of release, and we have to just implement a faster solution. I think that the problem is that management is not willing to see what people are actually doing during the work day and how much time doing something is actually taking. This leads to problems in the distribution of resources, and I think that has been our problem these days."

4.1.2 | Process divergence

Process divergence is an umbrella term, and according to Martini et al,[S4] "process designers create flawed processes, process executors diverge from well-designed processes, and the deficiencies of a supporting infrastructure might create difficulties." Thus, process divergence is not only itself a challenge, but it also leads to other issues such as not following standards in the highly regulated software domain, inappropriate planning and prioritization, and so on. The need for reworking, additional time, and concessions as well as drop in individual productivity all influence the quality of the software development process.[S3] Specifically, by not adhering to coding standards, junior developers expose themselves to accidental debt accumulation. Further, when product code reviews are not executed regularly, as defined by the process, debt is accumulated by the inclusion of a bad solution.[S16] In this context, process debt has both long-term and short-term consequences. In the long term, customer deliveries are delayed, software quality is poor, and other processes are impeded, whereas in the short-term, the immediate consequences are developer overload, errors, and a lowering of their morale (due to the tough job) as well as a lack of trust in the process.[S4]

4.1.3 | External dependencies

Formal ways of interacting with external parties, or a lack of them, might contribute to process debt. Yli-Huumo et al[S16] explained the problems of not *understanding customer expectations* and thus not being able to build customer satisfaction. Besides a process to understand the expectations from specific customers, more are needed to understand the market. If not, *a lack of knowledge about future changes* could cause an increase in process debt.[S16] Whereas a software development organization uses its own processes, other companies, units, teams, or special domains could all contribute to making existing processes suboptimal.[S4] These *special contexts and situations* could generate process debt and have costly consequences if they are not accounted for in the process.[S4] Moreover, both Martini et al[S4] and Ramač et al[S5] have noted the importance of infrastructure, in that *technology and tools* are important in terms of supporting the processes as well as automating and facilitating steps. More specifically, the inadequate choice of technology[S5] or lack of infrastructure[S4] could cause additional process debt, such as when an old configuration management tool does not support fast deliveries. In addition, *external trends* could also affect which processes are adopted in a company, and choosing those that are not suitable for the company can lead to process debt.[S4]

4.2 | People debt

People debt refers to issues with people that, if present in a software organization, can delay or hinder some development activities. An example of this kind of debt is too few people having the necessary expertise due to delayed training and/or hiring.[17]

People debt is a complex phenomenon focused on people-related issues with the close involvement of socio-organizational factors. The effect of this debt has a direct connection to productivity and people satisfaction.[S3] Our analysis identified four main themes of challenges regarding people debt, and they are summarized in Table 9.

4.2.1 | Lack of knowledge and experience

The key factor in people debt is the individual knowledge in an organization. In this context, the selected primary studies reported knowledge in terms of different issues, such as people's insufficient knowledge about technology, lack of product and context knowledge, lack of technical competencies, and/or hiring irresponsible staff. More specifically, there are several factors contributing to a *lack of product knowledge*. For example, *requirements from many different stakeholders* (e.g., *regulatory bodies)* or *rapid changes in technologies* make it difficult to identify which features a product should have after several years of its development.[S13] This lack of knowledge can lead to the requirements being considered vague or unclear.[S13]

Context knowledge is also a major issue in the embedded software area. For instance, original equipment manufacturers often do not provide all the details to protect their intellectual property.[S13] Additionally, when teams use an iterative development approach and information is not shared appropriately, individuals are unable to see the overall picture due to only being given part of the overall requirements in each iteration.[S13]

TABLE 9 People debt and causes

People debt theme	Causes	S2	S3	S4	S5	S7	S8	S10	S12	S13	S16
Lack of knowledge and experience	Insufficient knowledge		X		X						
	Lack of product knowledge				X					X	
	Lack of context knowledge				X					X	
	Lack of technical knowledge										X
	Lack of knowledge	X									
	Non-sharing knowledge				X						
	Lack of qualified professionals				X						
	Lack of experience				X						
	Cognitive distance							X			
	Newbie free-riding							X	X		
	Junior coders										X
Lack of commitment	Irresponsible staff or misconduct		X		X						
	Thinking in an old framework							X			
	Priggish members							X	X		
	Lack of commitment				X						
Inadequate management decision	Lack of gender diversity										X
	Team overloaded				X						
	Lack of opportunities	X									
	Lack of incentive	X				X					
	Lack of training OR lack of training opportunity	X			X						
	Inadequate management decision				X						
Morale	Developers' mistakes			X							
	Developers demotivated by some product defects				X	X					
	Demotivation of non-senior members								X		
	Lack of motivation				X						
	Psychological safety						X				

Further, *knowledge sharing* is one of the key elements of *social capital*, as software development involves people inside and outside their teams. In this regard, Tamburri et al[S10] noted that "in software engineering, cognitive distance can be thought of as the distance that developers perceive on the physical, technical, social and cultural level with respect to peers with considerable background differences."

Along with knowledge discussed above, another important factor is *hiring relevant and responsible people*. More specifically, Mandić et al[S3] found that a case company "[lost] a client due to people-related issues" and elaborated as follows: "We spent a lot of time and money on irresponsible people, and because of that, the project was stopped. We looked too long at short-term earnings and survival, instead of the long-term cooperation and quality." According to Ramač et al,[S5] lack of experience can lead to incurring debt and is likely to cause inappropriate planning, ill-defined processes, and ineffective project management. Further, new and junior coders accumulate unintentional debt mainly due to their lack of experience and are prone to cognitive distance.[S16]

4.2.2 | Lack of commitment

Regarding people debt, **misconduct** was reported as an issue in both Mandic et al[S3] and Ramač et al.[S5] Specifically, one type was hiring irresponsible people who lacked commitment to the project, whereas another was the presence of *priggish members*—those who make demands from others in an irritating manner.[S3, S12] They might pointlessly demand precise conformity or exaggerated propriety in a self-righteous manner, in particular. However, the software business requires a mindset of commitment towards innovation and being ready for change based on circumstances. Thus, people avoiding or refusing to work in a new way or consider innovative ideas is a sign of a *lack of commitment*.[S5] Here, a classic example is being stuck with a waterfall model despite the need for agile methods or DevOps.[S12]

4.2.3 | Inadequate management actions

This is a broader category in which the needs and requirements of teams and individuals are not recognized or ignored. Specifically, management does not offer opportunities to grow and/or training, ignores it when teams are being overloaded with work, and also does not provide incentives and take care of gender issues. Here, job-related training would improve technical skills, but other types of training, such as those involving team building and diversity, would help to mitigate gender issues in software teams.[S2] Moreover, Canedo et al[S2] reported that software development teams are mostly male dominant, and women face several challenges, such as the lack of acceptance of women in software development teams, sexual and moral harassment, lack of equality and gender diversity, and sexist and gender-related jokes. Here, the lack of trust, equal treatment, and incentives demotivate women. The same study also highlighted that there is a *lack of trust in women to accomplish some tasks*: "the most boring activities are always assigned to women because usually the men team-mates do not want to execute them."[S2] Further, management finds it more challenging to convince staff to have decent and neutral conversations among teams: "I had to adapt myself to the way the men talked, e.g., the slang and the bad words. When I arrived on the team, it was common for me to hear jokes from teammates that I was stupid because I am a woman."[S2]

4.2.4 | Morale

Morale is a multidimensional concept, which is defined by Peterson et al as follows: "[it is] a cognitive, emotional, and motivational stance toward the goals and tasks of a group. It subsumes confidence, optimism, enthusiasm, and loyalty as well as a sense of common purpose." Software development is a very complex process, and in today's world, it demands that developers understand their feelings and perceptions as well as the motivational factors in their environment. Specifically, there is a connection between developers' productivity and software quality and their satisfaction, happiness, and motivation.[S7] Additionally, software professionals' morale is affected by various factors such as the development process, quality of the overall software product, architecture, etc.[S7] In this context, TD harms morale, because developers see it as a roadblock to development and find that it makes it difficult for them to complete their tasks. On the other hand, TD management boosts employee morale.

TABLE 10 Social debt and causes

Social debt themes	Causes	S2	S5	S6	S8	S9	S10	S11	S17
Gender challenges	Gender bias	X							
	Lack of more inclusive work environments	X							
	Lack of trust that women could accomplish some tasks	X							
	Gender-equal communication	X							
	Lack of equal treatment	X							
	Lack of kindness	X							
Communication, collaboration, and coordination (3C)	Lack of communication (lack of communication depth, lack of communication latency, lack of feedback loops, lack of informal communication, lack of shared understanding, lack of unintentional communication)		X		X			X	
	Lack of collaboration ([unintentional] lack of conflict resolution), intensity of task-related collaboration, lack of consistency in employee availability, lack of knowledge transfer capabilities		X		X				
	Lack of coordination (lack of coordinative mechanisms, lack of formal control, lack of informal control), lack of leadership, lack of transparency, lack of team autonomy		X		X			X	
Community smells	Hyper-community						X		
	Power distance						X		
	Organizational silo					X			X
	Black cloud					X			X
	Lone wolf			X		X			
	Radio silence					X			X

The key reason is that it is linked to positive personal and interpersonal feedback, which allows developers to perform their tasks in a better manner and increase software quality in the future.[S7] Dreesen et al[S8] also highlighted that psychological safety plays an important role in software development, as the lack of it contributes to various types of debt collection, such as social debt.[S8]

4.3 | Social debt

Social debt is analogous to technical debt in many ways: It represents the state of software development organizations as the result of "accumulated" decisions. In the case of social debt, decisions are about people and their interactions.[S17]

Tamburri et al[s7, s8, s19, s23, s29] were the first to extensively investigate social elements of debt under the term "social debt" in software development. First, it should be noted that software development is a sociotechnical process. Thus, social debt is closely linked to TD, as, for example, uninformed sociotechnical decisions compound both social and technical debts that cannot be "paid back" easily.[S17] Second, social debt involves people as well as their interactions and related decisions. Third, awareness maintenance methods are conceptually similar to the concept of social debt, because they track and preserve project knowledge to reduce delays and associated debt.[S17] Fourth, *social debt is a cumulative and increasing cost in the current state of things*, *connected to invisible and negative effects within a development community*.[S17] Thus, it easily emerges in the scaling of agile methods or offshoring, and sociotechnical congruence is the first rudimental sign of the same.[S17] In this context, Tamburri et al[S17] presented numerous aspects of social debt, such as how it is indirectly related to socio-technical decisions and how it is an emerging attribute of the development community. Several triggers result in the creation of social debt, such as omissions, compromises, bad behavior, not seeking or giving help to colleagues, power distance, gender bias, etc.[S2, S8] The complete list of causes of this debt is provided in Table 10.

4.3.1 | Gender challenges

In general, the presence of female workers in the software business is unbalanced, as compared with men, fewer women work in the information and communication industry. The issue of gender is Pandora's box, but in this paper, we only focus on it in the context of software development teams and their connections to social debt. The existing literature has highlighted that gender-diverse organizations have higher levels of innovation, problem-solving skills, and a healthier work environment.[S2] However, Canedo et al[S2] reported several issues, for example, the lack of trust in women's competencies and capabilities as well as equal treatment and communication, harassment, etc. Men are also likelier to hold technical leadership positions in software development teams. In this context, the lack of trust can be better expressed using this example: "the most boring activities are always assigned to women because usually the men team-mates do not want to execute them."[S2] Further, compared with men, women are treated differently in teams: "when the team is assembled by a woman, the team behaves differently. Also, one woman in a leadership position sometimes still causes strangeness in other teams."[S2]

Moreover, the most recurring barrier is related to the *lack of gender-equal communication*, as some men do not talk equally to their female teammates. In order to express an opinion, women need to interrupt men's speech; this happens even if the person who wants to talk is a female project manager.[S2] One interviewee also reported that the following: "I had to adapt myself to the way the men talked, e.g., the slang and the bad words. When I arrived on the team, it was common for me to hear jokes from teammates that I was stupid, because I am a woman."[S2]

4.3.2 | Challenges in communication, collaboration, and coordination

Communication, collaboration, and coordination are sometimes referred to as the 3Cs.[S8] First, in terms of communication, social debt can occur due to omissions, compromises, or bad behavior.[S5, S8, S11] Moreover, the concept of this form of debt shares some similarities with that of *social capital*, which refers to resources within and derived from social relationships among actors.[S8] Building social capital can be seen as a process of building attributes such as trust, relationships, and a shared understanding. A related concept is *psychological safety*, which refers to the perceived climate and the perception's effect on the actors' resulting behavior.[S8] The two concepts of social capital and psychological safety are interconnected, but they consider different aspects: the former is focused on individual actors and their place within the social network, whereas the latter is more concerned with the perceptions of the team.[S8] In the context of this paper, if developers use their social capital to force omissions in the general communication patterns or to coerce compromises, the results may lead to social debt and undermine the positive influences that enabled the building of the capital in the first place.[S8] For example, *a lack of communication depth* could be due to remote communication in which gestures and physical features cannot be seen or are not as prominent.[S8] Another possible problem due to digital meetings could be a *lack of communication latency* because of slower or staggered communication.[S8] Besides meetings, the *lack of informal communication*, such as non-structured, ad hoc talks, could add to social debt.[S8] Less spontaneous visits to colleagues' offices or canteens are also examples of a *lack of unintentional* *communication*.[S8] In addition, the *lack of feedback loops* could mean less or missing business-related information, and the consequent misinterpretations could also lead to a *lack of shared understanding*.[S8]

Second, collaboration challenges are also possible reasons for the building up of social debt.[S5, S8] Here, a *lack of conflict resolution* could occur, sometimes unintentionally, if conflicts are less frequently addressed or resolved.[S8] In an environment with a need for highly task-oriented interactions, the *intensity of task-related collaboration* could be a possible problem.[S8] However, according to a study by Dreesen et al,[S8] the aforementioned intensity was the only investigated factor that had both a promoting and mitigating effect on social debt. Additionally, if the availability of team members is not transparent and/or synchronized regularly, there could be a problem in terms of the *lack of consistency in employee availability*.[S8] In situations where knowledge transfer is reduced or postponed[S8] or documentation is lacking,[S5] a *lack of knowledge transfer capabilities* could also cause issues.

Finally, coordination entails several factors that possibly cause social debt.[S5, S8, S11] When rules and schedules for teamwork have not been set up (e.g., deciding on and sticking to core working hours) or are loosened, there is a *lack of coordinative mechanisms*. If the behavior of team members cannot be observed and the desired conduct cannot be identified—for example, if the team is not collocated—the *lack of formal control* could cause problems. However, a *lack of informal control* could also cause problems if values and norms that promote teamwork are not in line with a company's goals.[S8] One example of such values and norms could be, as Martini et al[S11] pointed out, a *lack of team autonomy* if the company does not allow a mandate for making decisions. Moreover, if common leadership tasks are neglected, such as inspiring, encouraging, and supporting the team[S8] or inappropriate planning,[S5] the *lack of leadership* could increase social debt. Further, if the behaviors of individuals or teams are unclear or missing, there can also be a *lack of transparency*.[S8]

4.3.3 | Community smells

The sum of negative and unforeseen costs/consequences given by suboptimal social relations among developers has been named "community smells."[S9] According to Catolino et al,[S9] these smells represent one of the main causes of social debt and, as such, threaten the entire management of software systems, because they may substantially decrease the level of trust within a community.

There is the first type of community smell, called *hyper-community*,[S10] which refers to a community where decisions are made based on groupthink and are thus biased and could cause social debt. Such a hyper-community could, in turn, also influence other subcommunities in its fold.[S10] The second smell, called *power distance*, takes place when less powerful or responsible members of a software development community perceive, accept, and/or expect a distance from powerholders.[S10] If there is a presence of siloed areas of the developer community that do not communicate with the rest of the community, except through one or two of their respective members, it is a sign of what is called the "organizational silo effect."[S9, S17] Signs of this smell include the high decoupling between tasks and a lack of communication.[S17] Meanwhile, information overload due to a lack of structured communications or cooperation governance is a smell caused by the "black cloud effect,"[S9, S17] which is defined as a lack of boundary spanners and a proper sharing of initiatives and protocols.[S17] Additionally, another smell, called the "lone wolf effect,"[S9, S6] could also appear when the development community has unsanctioned or defiant contributors who carry out their work with little consideration for their peers, decisions, and communication. An example of this smell is when non-architects are forced to make decisions because actual architects are "too few and far apart."[S6] When a highly formal and complex organizational structure full of "regular procedures" forces changes to be slowed down and consequently loses time between hierarchical people,[S17] the smell is called the "radio silence effect."[S9, S17] A sign of this smell is that one member interposes himself/herself into every formal interaction across two or more subcommunities with little or no flexibility to introduce other parallel channels.[S9]

5 | PREVENTION STRATEGIES FOR NTD

This section presents prevention strategies and actions to mitigate the three types of NTD (i.e., process, people, and social debts).

5.1 | Prevention of process debt

Regarding organizational challenges, Tamburri et al[S12] investigated architecture issues causing process debt and presented a prevention strategy called the "Dev-Ops shift-left." This shift is explained as moving the development-to-operations team to an earlier ("left") position in the development life cycle, which involves architecture and development, and thus having operations staff also work as developers. These authors also suggested as prevention strategies as follows: designing a people-oriented rather than feature-oriented architecture and a model-based mediation of knowledge and keeping track of actions, agreements, and expectations through work-division artefacts.[S12] Meanwhile, to prevent a mismatch between requirements at different abstraction levels, Liebel et al[S13] suggested documenting decision rationales—that is, the why and how a

TABLE 11 Process debt prevention strategies

Process debt theme	Process debt causes mitigate	Preventive actions	Source
Organizational challenges	Lack of engagement in development	A DevOps shift-left approach to address issues earlier during development and having operations staff also be developers	S12
		Designing a people-oriented rather than feature-oriented architecture	
	Business decisions	Communication structure between business management and development team	S16
	A mismatch between requirements on different abstraction levels	Document decision rationales (*why* and *how* a requirement was broken down to a more detailed level)	S13
	Excessive conformity to standards	Open communication and informality and model-based mediation of knowledge	S12
	Lack of innovation	Daily stand-ups, and keeping track of actions, agreements, and expectations through work-division artefacts	S12
Process divergence	(Not presented)	Following the project planning	S1
		Following a well-defined project process	
		Refactoring	
		Risk and impact analysis	
		Well-defined requirement	
	Lack of coding standards	Coding standards and guides	S16
External dependencies	None	None	None

requirement should be broken down to a more detailed level. Additionally, Freire et al[S1] presented five strategies to prevent process divergence in agile, hybrid, or traditional software engineering: following a well-defined project process, adhering to project planning, refactoring, performing risk and impact analysis, and having well-defined requirements. The authors went on to suggest two criteria for choosing a prevention strategy: (i) consider the most commonly used preventive actions as a starting point and (ii) combine preventive actions based on debt types. Finally, Yli-Huumo et al[S16] suggested the creation of coding standards and guides as well as improving the communication structure between business management and development teams.

Table 11 shows a summary of prevention strategies to mitigate process debt for a few of the identified causes of organizational challenges and process divergence, as none of the papers have suggested strategies to manage external dependencies.

5.2 | Prevention of people debt

For the category called the "lack of knowledge and experience," Liebel et al[S13] suggested the use of graphical models as "thinking tools" to overcome the inadequacy of product knowledge. Regarding the lack of context knowledge, these authors suggested increased traceability between requirements. In particular, if a clear link is provided to a higher level of abstraction, it is possible for developers to better understand the context.[S13] Meanwhile, Freire et al[S1] proposed the improvement of technical knowledge by training and good communication to build an effective team. As onboarding of new employees is important and to ensure that the problems of "newbie free-riding" are mitigated, Tamburri et al[S12] suggested several preventive actions, one of which is called "mood polling." This entails sharing your mood as a team member regarding the organizational scenario and its mapping onto architecture elements. By investigating reported "bad moods," developers are consequently empowered to deal with architectural problems. Further, Tamburri et al[S12] explained that harmonizing responsibilities by implementing roles and practices from Scrum or other agile frameworks could minimize problems with priggish members. However, regarding inadequate management decisions, only a prevention strategy concerning the lack of gender equality has been addressed in the literature. Specifically, Canedo et al[S2] proposed programs to promote gender diversity by using campaigns and/or lectures on the importance of having gender diversity, hiring more women, giving more opportunities to women, and increasing women's representation in the organization.

Table 12 shows a summary of prevention strategies to mitigate people debt for a few of the identified causes on the lack of knowledge and experience as well as commitment and inadequate management decisions. This is because none of the papers have suggested prevention strategies to manage morale issues.

TABLE 12 People debt prevention strategies

People debt theme	People debt cause to mitigate	Preventive actions	Source
Lack of knowledge and experience	Lack of product knowledge	Graphical models as a "thinking tool" to better understand the system	S13
	Lack of context knowledge	Increased traceability between requirements, especially to higher levels of abstraction	S13
	Lack of technical knowledge	Improve the technical knowledge by training and good communication	S1
	Newbie free-riding	Coordination management and engagement creation	S12
		Architect-as-a-coach	
		Explicit empowerment of architecture decision changes	
		Organizational monitoring—for example, anonymous mood polling	
Lack of commitment	Priggish members	Harmonizing responsibilities	S12
Inadequate management decision	Lack of gender diversity	Programs to promote gender diversity	S2
Morale	None	None	None

5.3 | Prevention of social debt

Canedo et al[S2] suggested gender diversity programs, as noted above, to mitigate the lack of gender diversity and also to prevent gender challenges. Besides the areas covered above, these gender diversity programs should include policies against harassment and define those for the inclusion of women as well as allow for the sharing of experiences and the curbing of sexist attitudes.[S2]

Additionally, Dreseen et al[S8] presented a prevention strategy containing four intertwined preventive actions to manage the lack of communication and collaboration. They explained that "honest and open communication" along with "reduced upfront costs for initializing interaction and communication" contribute to an environment in which social conflicts may be addressed and solved more easily; therefore, social debt can be decreased. They also found that the action of "improving psychological safety" had a mitigating effect on both lack of communication and collaboration. Meanwhile, the "intensity of task-related collaborations" had both a promoting and mitigating effect.[S8]

Overall, most of the prevention strategies put forward in the reviewed primary studies focus on mitigating community smells. In terms of hyper-community smells, where groupthink could cause social debt, Tamburri et al[S12] proposed an architect-as-coach initiative where architects incite doubt through discussion and reverse logic. Next, Catolino et al[S9] proposed six strategies to mitigate organizational silo smells, due to which siloed areas of the developer community do not communicate, as well as radio silence smells, where delays in information sharing could cause social debt: (i) restructure the community; (ii) create a communication plan; (iii) organize mentoring; (iv) use exercises on cohesion; (v) monitoring; and (vi) introduce a social-rewarding mechanism. For the organizational silo smells, Tamburri et al[S17] suggested adopting a "social wiki" where practitioners' profiles are combined with the artefacts under their care and the connected documentation, whereas for the radio silence smells, they suggested establishing an online learning community as a prevention strategy.

Meanwhile, Catolino et al[S9] suggested three strategies to mitigate the black cloud smell, due to which information overload could cause social debt. However, these suggestions have already been proposed to manage the previously mentioned organizational silo smell: restructure the community, create a communication plan, and introduce a social-rewarding mechanism. Regarding this smell as well, Tamburri et al[S17] again suggested adopting a "social wiki" as a prevention strategy. Finally, Catolino et al[S9] suggested five strategies to mitigate the lone wolf smell, due to which defiant contributors who carry out their work with little consideration for their peers could cause social debt. These strategies are all the same as those suggested to prevent organizational silo smells except for "create a communication plan." Here, Tamburri et al[S6] suggested the use of architecture boards consisting of software architects and other interested parties to mitigate the lone wolf smells.

Table 13 shows a summary of prevention strategies to mitigate social debt. Several prevention strategies that have been explained are proposed for community smells, but there are only a few suggestions to mitigate gender challenges and 3C challenges.

6 | DIRECTIONS FOR FUTURE RESEARCH

The focus of this mapping review was to provide an overview of what has been investigated regarding NTD in scientific literature. We found that most of the research focuses on TD and less attention has been paid to NTD. However, it should be noted that NTD are factors that contribute to TD. This means that the latter is not always based on technical aspects and might be caused by other factors such as various stakeholders, adopted processes, and social and cultural relatedness, which are less investigated in this context. In light of our analysis and synthesis of the previously identified primary studies, the following governing questions should be examined and researched in the future:

- Are software professionals familiar with the concept of NTD?
- What similarities and differences exist between TD and NTD?
- Is there any connection between sustainability debt and NTD?
- What are the factors that affect process, social, and people debts during the software life cycle? Are there patterns in the behaviors of such factors?
- What effects do NTD have on software projects?
- What types of events characterize the causes and effects of process, social, and people debts?
- How do process, social, and people debts originate and evolve in software development settings?
- What are the drivers of NTD in large-scale distributed agile software development teams?
- What are the factors mitigating the build-up of NTD in large-scale distributed agile software development teams?
- What types of NTD are elicited and discussed at the team and inter-team levels in large-scale agile software development teams?
- How do process, social, and people debts affect teamwork in software development teams?
- To what extent do process, social, and people debts represent threats to team cooperation?
- To what extent do practitioners (or managers) allow for the existence of NTDs that affect cooperation in their organizations?
- Is there any connection between interdisciplinary requirement engineering and NTD accumulation?
- Does organizational structure have any effect on NTD accumulation?
- Does a well-defined process and planning prevent process debt in both collocated and distributed large-scale agile software development projects?
- Does the pressure of deadlines contribute to NTD?
- What are the effects of business decisions on NTD?
- Does the communication structure between business management and the development team have an impact on NTD accumulation?
- Does gender imbalance in teams have any effect on NTD accumulation in software projects?
- Does the occurrence of NTD influence professional morale in the software industry?
- How do social elements (e.g., personal traits, self-efficacy, social identity, social ties, and inter- and intra-team interaction) play a role in NTD accumulation?
- How does NTD influence developers' psychological and motivational states in terms of their software development?
- To what extent can social and people debts affect the enforcement of the ACM code of ethics and professional conduct?
- What are the potential ethical and moral issues that software engineering professionals may face due to social and people debts? What is the knowledge-building role of the leader in NTD accumulation or management?
- What are the strategies applied by practitioners to deal with the causes of process, social, and people debts?
- How do stakeholders influence NTD and its management in organizations?

Thus, as seen above, there are many open research questions. However, it is unclear as to what types of mitigation strategies are necessary to handle NTD throughout an organization and what strategies have to be adopted. Moreover, because the new normal caused by the COVID-19 pandemic has been causing significant changes in numerous software and services businesses, it is worthwhile to investigate work satisfaction and NTD in both co-located and large-scale agile software development projects. It would also be interesting to investigate NTD under the lens of established theories such as the social capital, psychological safety, or control theories as well as communication, collaboration, and cooperation challenges in connection with the *media naturalness theory*. Additionally, Liebel et al[S13] presented a three-point agenda for future debt research: (i) *the need for a process that allows for sufficient levels of uncertainty during early requirement engineering*; (ii) *the need for an organizational structure that effectively supports interdisciplinary requirement engineering while taking the central role of software into consideration*; and (iii) *the need for concepts as well as organizational structures that allow for and support the management of "requirements debt."* Further, in terms of software processes, the value for stakeholders is always considered, as ill-designed ones diverge from those that are well designed, and the lack of a supporting environment results in waste. These phenomena require investigations of their own.

7 | DISCUSSION

The topic of debt is relatively new as compared with other domains, such as software quality and testing. However, although there are significant studies published on TD, NTD has remained less explored. Among the different types of debt proposed in the work by Lenarduzzi et al,[5] we chose to investigate three types of debt under the umbrella of NTD: process, people, and social debts. In this study, we then adopted a systematic

TABLE 13 Social debt prevention strategies

Social debt theme	Social debt cause to mitigate	Preventive actions	Source
Gender challenges	Gender bias	Programs to promote gender diversity among employees	S2
	Lack of more inclusive work environments		
	Lack of trust that women		
	Gender-equal communication		
	Lack of equal treatment		
	Lack of kindness		
Communication, collaboration and coordination challenges	Lack of communication	Honest and open communication, intensity of task-related communication, improving psychological safety	S8
	Lack of collaboration	Intensity of task-related collaboration, reduced upfront costs for initializing interactions or communications, improving psychological safety	S8
Community smells	Hyper-community	Architect-as-a-coach	S12
		Inciting doubt through discussion and reverse logic	
	Organizational silo smell	Restructure the community	S9
		Create communication plan	
		Mentoring	
		Cohesion exercising	
		Monitoring	
		Introduce a social-rewarding mechanism	
		Adopt a “social wiki”	S17
	Black cloud smell	Restructure the community	S9
		Create communication plan	
		Introduce a social sanctioning mechanism	
		Adopt a “social wiki”	S17
	Radio silence smell	Restructure the community	S9
		Create communication plan	
		Mentoring	
		Cohesion exercising	
		Monitoring	
		Introduce a social sanctioning mechanism	
		Learning-community	S17
	Lone wolf smell	Restructure the community	S9
		Mentoring	
		Cohesion exercising	
		Monitoring	
		Introduce a social sanctioning mechanism	
		Architecture boards	S6

mapping review approach to identify, classify, and analyze 175 studies on NTD published between 2000 and 2021 in the field of software engineering. Of these, 17 articles were identified as primary studies, which were chosen based on the selection criteria defined for this review: those with acceptable academic rigor, credibility, and relevance.

Overall, we found that in recent years, there has been an increase in studies in the NTD research area. Moreover, the primary studies predominantly used a mixed-method approach ($n = 9$), whereas qualitative ($n = 4$) and quantitative ($n = 4$) approaches were adopted in four papers each. The mixed-method approach is interesting and able to offer valuable insight, but the progression of the used case studies was not explicit (e.g., when and how NTD emerged and had its effects). Further, as the NTD research area is still in its infancy, this mapping study did not find any longitudinal studies on this topic. However, the realities of NTD have been reported as anecdotal evidence. In terms of defining various forms of NTD, such as social, process, and people debts, this mapping study identified a lack of cohesion. This raises the concern that in the long term, NTD in software engineering could lack the cumulative building of knowledge or that there will be "fragmented adhocracy."

NTD has a similar analogy to TD in terms of principles and interest. More specifically, the process debt metaphor supports the prioritization of process improvements. Examples of process debt are insufficient processes and documentation, inadequate software engineering practices, tools and planning, etc. This can be implied, as managing process debt contributes to the management of TD. In addition, there are techniques to help prioritize the aforementioned debt over other competing activities such as features, bug fixing, TD refactoring, etc. Such strategies are appealing, but they require adaptation and research to confirm or demonstrate their application in various contexts.

In terms of software processes, the value for stakeholders is always considered, and ill-designed ones diverge from those that are well designed, and the lack of a supporting environment results in waste. Overall, there are three main patterns in the occurrence of process debt: bad or flawed process design, divergence from a well-designed process, and lack of supporting infrastructure. In the case of process debt assessment, it is important to take the value of different stakeholders into consideration and pay attention to the process design. An ill-designed process leads to initiation waste and has long-term negative effects, such as the costs of making changes. Moreover, in the process debt, people are a crucial factor. The exploration of process competencies is critical in projects where software engineering teams work closely with mechanical and electrical engineers. More specifically, software and hardware engineers have different vocabularies, making it difficult to communicate with each another and causing problems in cross-functional teams. This is because processes that work well in one domain or context may not work well in another. In this context, the systematic research of metrics reflecting software processes and examining product characteristics across a wider range of situations (e.g., in industrial as well as hybrid open-/closed-sources) would be valuable. According to Freire et al,[S1] following a well-defined project process and plan prevents process debt. However, it requires investigations of both collocated and distributed large-scale agile software development projects. Nevertheless, introducing such processes in a system engineering context is not easy due to the lack of interdisciplinary understanding. This is another avenue for people and social debt-related research, under one umbrella, on how teamwork can be harmonized and how people from two different domains can be better understood as well as communicate with each other.

Meanwhile, people debt is more related to people-centric issues such as the lack of specific skills and knowledge among workers, hiring of irresponsible staff, and unclear quality-related roles and responsibilities as well as workers' psychological aspects, morale, and emotions. Moreover, people debt has a direct connection to TD and affects individual productivity and satisfaction. Specifically, developers' morale and productivity go down with TD, which further hinders individual performances, motivation, progress, and the achievement of their goals.[S4, S15, S16] In addition, women face gender bias in software development teams, as highlighted by Canedo et al,[S2] as well as impediments such as a lack of trust, credibility, confidence, and recognition. Team cohesion is harmed by such biases, which can lead to the buildup of various types of debt. To confirm and augment these findings, gender imbalance in software development teams should be investigated in other cultures as well. Thus, future studies in this area may help software development teams to reduce social and people debt.

The lack of qualified professionals and the hiring of irresponsible staff are contributors to short- and long-term debt in software projects. Meanwhile, in social debt, the decisions are more about people and their interactions, which are intertwined with TD. This indicates that bad sociotechnical decisions generate both technical and social debts. An example of such a decision would be the incremental contrariness of the addition of "technical experts" to teams in the development community or that of consultants to help with product renewal. Further, the causes of social debt have similarities with that of the 3C model of communication, the activities of which are crucial for software production and related operational activities. Specifically, communication is critical to the realization of organizational activities and the understanding of tasks, the coordination of activities with different stakeholders contributes to the efficient achievement of business goals, and cooperation on given tasks is necessary to handle concurrent and shared expertise. Naturally, various challenges arise in software development teamwork due to hindrances in communication and the lack of cooperation and coordination. However, to minimize social issues, organizations need to undertake trial-and-error exercises. Additionally, it is also evident that NTD and TD go hand in hand. Thus, we urge practitioners to pay attention to NTD as much as other types of TD.

Through this study, we have identified several preventative strategies and actions suggested for mitigating NTD, but they are few in number as compared with the identified causes and consequences. For process debt, prevention strategies are only suggested for a few of the identified causes of organizational challenges and process divergence. Moreover, none of the examined studies suggested prevention strategies to manage external dependencies. In investigating people debt prevention, we found only a few preventative measures targeting the identified causes of the lack of knowledge, experience, and commitment as well as inadequate management decisions. Here, none of the selected papers suggested prevention strategies to manage morale issues. In preventing or mitigating social debt, several strategies were proposed to manage community smells, but only a few were proposed to mitigate gender and 3C challenges. Meanwhile, Eduardo summarized and highlighted the frameworks (i.e., CAFFEA framework, architectural tactics, and DAHLIA) and tools (GEEZMO, CodeFace4Smell, YOSHI) that can be used to handle social debt.[30] This mapping of causes and prevention strategies displays the underdeveloped state of NTD research, as the few suggested prevention strategies have not been tested or evaluated in any scientific sense.

7.1 | Implications for research

This systematic mapping study has several implications for research and practice. In terms of research, our review shows a clear need for more empirical studies on NTD. Only seven of the identified papers on NTD were published in academic journals, and none of the primary studies received full points based on our quality assessment framework. This means that there is a lack of empirical studies with a high level of evidence that can help in understanding more about NTD and how the negative effects caused by it can be prevented.

Moreover, prevention strategies and actions should be significantly refined, although more empirical studies are necessary to provide evidence of helpful actions for NTD mitigation. Such strategies for many of the identified debts have neither been expressed nor suggested in the primary studies. In addition, they have often only been mentioned in the related primary studies without a deeper discussion or investigation into their practical usage or efficiency. Further, as debt is not an "island" and can affect or be affected by other types of debt, the relationships between different types of NTD should also be further investigated.

The mapping study also showed that a range of research methods have been applied in the selected studies. However, it is necessary to employ both flexible and fixed research designs if a deeper understanding of NTD is to be gained. Meanwhile, Edmondson and McManus[31] argued that the research design needs to fit the current state of theory and research, which they went on to divide into the following three categories: nascent, intermediate, and mature. Regarding NTD, we believe the current state of theory and research on methods is clearly nascent, which suggests the need for exploratory qualitative studies as per the study cited above.[31]

7.2 | Implications for practice

For practitioners, this mapping study identifies many promising studies that show how prevention strategies and actions have been used to mitigate the negative effects of NTD. However, prevention strategies for many of the identified debts have not been suggested, so additional industrial cases are needed to show how the different types of NTD should be dealt with in practice. These cases may increase the awareness of different types of NTD and might inspire different stakeholders to consider its mitigation more seriously. Thus, we would like to urge companies to participate in research projects in the future in order to target research goals, regarding the prevention and mitigation of NTD, that are relevant to the software industry.

8 | VALIDITY THREATS AND LIMITATIONS OF THE STUDY

Every study faces threats to its validity in one form or another. In this mapping study, we followed the validity framework by Wohlin et al[32] and Unterkalmsteiner et al.[33] In this context, *construct validity* refers to using the right measures for the concept being studied.[26,32] Here, to minimize construct validity, the review was systematically designed, inclusion and exclusion criteria were applied, and the data extraction of the primary studies was recorded. To further reduce this threat, the senior colleagues in the software engineering department validated the study protocols. Meanwhile, *external validity* refers to the extent to which the study results are generalizable.[26,32] This threat was minimized by following the rigorous research methodology put forward by Petersen et al[26] and Dybå and Dingsøyr.[19] Next, *internal validity* relates to causal relationships. However, as we did not investigate statistical causal relationships in terms of NTD, this type of validity is not considered a threat to this study. Finally, *conclusion validity* refers to researchers' bias in the interpretation of the data. Although this cannot be fully eliminated, we took several steps to minimize it. For example, two researchers were involved in the mapping investigation, and a full "audit trail" was maintained, starting from database searches till the obtaining of primary studies. In addition, two researchers independently analyzed the primary studies and later discussed them in workshop settings to conclude the synthesis and reporting.

Publication bias: Systematic reviews have a common bias, as positive outcomes are more published compared with negative ones.[27] This bias can be observed in the primary studies in this review, as NTD challenges have been reported with few solutions. One of the challenges with reporting NTD is its multidimensional nature, which consists of both social and technical aspects. We consider this threat as orthogonal to the aim of our mapping study, which is to present the current state of NTD research. Furthermore, publication bias can also be linked to the sources and publication channels of data in a study. In this review, we used five prominent databases (i.e., ACM, SpringerLink, Web of Science, ScienceDirect, IEEE Xplore, and Scopus), which resulted in access to most publications, and these have also been previously used in similar types of reviews in software engineering.[28]

Identification of primary studies: We piloted the search string and adopted a strategy to focus on key NTD types. We avoided the single use of the term "debt" because it is very broad in meaning and yields noise and an unmanageable number of irrelevant results from various disciplines, such as finance. Consequently, we used focus terms such as "process debt" and "people debt" to construct search strings with the help of library staff and software engineering researchers. The flip side of this strategy was that we received a limited number of hits, but these were more relevant. However, we observed high inconsistency throughout the analysis and thus filtered the retrieved studies based on term usage of NTDs.

Some studies reported it as an independent topic or phenomenon, whereas others placed it under various umbrella terms such as TD. For example, Martini et al[S4] covered process debt, while Tamburri et al[S17, S6] clearly focused on social debt. Additionally, none of the studies used the term "nontechnical debt," whereas a few others mentioned various types with their exact names such as social and process debts.

Study selection and data extraction: Systematic review protocols have to be prepared upfront, and this includes the definition of the inclusion/exclusion criteria. This helps mitigate study selection and data extraction threats. The next threat to be considered is researchers' personal bias. Based on the technique suggested by Kitchenham and Charters,[27] we reduced such biases by using a rigid protocol for study selection as well as by having two researchers independently filter the primary studies. Next, for the quality assessment, they independently applied the criteria proposed by Dybå and Dingsøyr.[28] Later, the two researchers used multiple sessions to discuss their conflicts and mutually adjusted each other's biases.

9 | CONCLUSION

With this paper, we have conducted the first systematic mapping study to provide a structured understanding of the current state of social, process, and people debts in software engineering research. This was achieved by identifying 17 primary studies out of 175 related to the aforementioned debts published in the last two decades. The primary studies were then analyzed based on the following: (i) frequency of publication by year; (ii) publication channels; (iii) research method; (iv) quality; (vi) definitions of various NTDs; and (v) various debt causes, prevention strategies, and future research agendas.

The mapping showed that social and process debt research is in the initial stages as compared with the topic of TD. Meanwhile, as compared with the aforementioned debts, social debt has been predominantly studied. Generally, the majority of studies examined in this paper adopted a mixed-method approach and reported results in single case studies. However, there are no long-term or longitudinal studies yet that have reported the causes and effects of NTD in software development. Nevertheless, it is clear from the studies that NTD contributes to the accumulation of TD and has a negative effect if ignored in companies. Although most of the research on NTD is at an early stage and the results are anecdotal, it has been gaining momentum with the publishing of significant studies since 2019.

Overall, companies need to manage many technical and NTD issues in software projects. Sometimes, these issues are inevitable and multidimensional due to business uncertainties or internal and external environmental factors. Moreover, besides TD, other issues contribute to the accumulation of various types of debt and negatively affect software life cycles. Therefore, it is important to understand these debts to minimize their detrimental effects. Although this study offers a range of NTD mitigation strategies, it does not guarantee their success. Thus, the plethora of mentioned strategies needs to be experimented with and might also require other supporting practices based on the context. Nevertheless, research on NTD is largely unexplored, and therefore, the research community needs to provide a contemporary perspective on different types of NTDs (i.e., social, process, cultural, and people debts). Additionally, there is also a need to not just study TD in isolation but also include NTDs that play critical roles in software and system development in general. Moreover, future research should take the following into consideration: (i) There is a need to increase the number of rigorous empirical studies on NTDs; (ii) how the principles and interests of NTDs are to be systematically evaluated and measured; (iii) studies are required to be more explicit regarding their validity threats and mitigation strategies; and (iii) cumulative knowledge needs to be built on. With this in mind, we also aim to develop a taxonomy of NTD and a guiding framework to assist in its mitigation in software projects.

ACKNOWLEDGEMENTS

This publication was made possible by the Knowledge Foundation Sweden Grant and Karlstad University, Sweden.

DATA AVAILABILITY STATEMENT

Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

ORCID

Muhammad Ovais Ahmad https://orcid.org/0000-0002-7885-0369
Tomas Gustavsson https://orcid.org/0000-0002-1512-6592

REFERENCES

1. Dikert K, Paasivaara M, Lassenius C. Challenges and success factors for large-scale agile transformations: a systematic literature review. *J Syst Softw*. 2016;119:87-108. doi:10.1016/j.jss.2016.06.013
2. Ahmad MO, Dennehy D, Conboy K, Oivo M. Kanban in software engineering: a systematic mapping study. *J Syst Softw*. 2018;137:96-113. doi:10.1016/j.jss.2017.11.045
3. Cunningham W. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger*. 1992;4(2):29-30. doi:10.1145/157710.157715
4. Avgeriou P, Kruchten P, Ozkaya I, Seaman C. Managing technical debt in software engineering (dagstuhl seminar 16162). In Dagstuhl Reports 2016 (Vol. 6, No. 4). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

5. Lenarduzzi V, Besker T, Taibi D, Martini A, Fontana FA. A systematic literature review on technical debt prioritization: strategies, processes, factors, and tools. *J Syst Softw*. 2021;171:110827. doi:10.1016/j.jss.2020.110827
6. Alfayez R, Alwehaibi W, Winn R, Venson E, Boehm B. A systematic literature review of technical debt prioritization. In Proceedings of the 3rd International Conference on Technical Debt. 2020:1-10.
7. Codabux Z, Williams BJ, Bradshaw GL, Cantor M. An empirical assessment of technical debt practices in industry. *J Softw Evol Proc*. 2017;29(10):e1894.
8. Allman E. Managing technical debt. *Commun ACM*. 2012;55(5):50-55. doi:10.1145/2160718.2160733
9. Krasner Herb. The cost of poor software quality in the US: a 2020 report. Published Online 2021. https://www.it-cisq.org/pdf/cpsq-2020-report.pdf
10. Denmark. (2017). Danish Ministry of Finance: Regeringens kasseeftersyn på it-området. Copenhagen.
11. Sweden. (2019). The Swedish National Audit: Föråldrade it-system–Hinder för en effektiv digitalisering.
12. Wilson C, Mergel I. Overcoming barriers to digital government: mapping the strategies of digital champions. *Gov Inf Q*. 2022;39(2):101681. doi:10.1016/j.giq.2022.101681
13. Avgeriou P, Ernst NA, Nord RL, Kruchten P. Technical debt: broadening perspectives report on the seventh workshop on managing technical debt (MTD 2015). *ACM SIGSOFT Softw Eng Notes*. 2016;41(2):38-41. doi:10.1145/2894784.2894800
14. Klinger T, Tarr P, Wagstrom P, Williams C. An enterprise perspective on technical debt. In Proceedings of the 2nd Workshop on managing technical debt; 2011:35-38.
15. Betz S, Becker C, Chitchyan R, et al. Sustainability debt: a metaphor to support sustainability design decisions; 2015.
16. Kruchten P, Nord RL, Ozkaya I. Technical debt: from metaphor to theory and practice. *IEEE Softw*. 2012;29(6):18-21. doi:10.1109/MS.2012.167
17. Kalenda M, Hyna P, Rossi B. Scaling agile in large organizations: practices, challenges, and success factors. *J Softw Evol Proc*. 2018;30(10):e1954.
18. Tamburri DA. Software architecture social debt: managing the incommunicability factor. *IEEE Trans Comput Soc Syst*. 2019;6(1):20-37. doi:10.1109/TCSS.2018.2886433
19. Yli-Huumo J, Maglyas A, Smolander K. The effects of software process evolution to technical debt—perceptions from three large software projects. In: *Managing Software Process Evolution: Traditional, Agile and Beyond - How to Handle Process Change*. Springer; 2016:305-327. doi:10.1007/978-3-319-31545-4_15
20. Khomyakov I, Makhmutov Z, Mirgalimova R, Sillitti A. Automated measurement of technical debt: a systematic literature review. In ICEIS 2019-Proceedings of the 21st International Conference on Enterprise Information Systems; 2019:95-106.
21. Rios N, de Mendonça Neto MG, Spínola RO. A tertiary study on technical debt: types, management strategies, research trends, and base information for practitioners. *Inf Softw Technol*. 2018;102:117-145. doi:10.1016/j.infsof.2018.05.010
22. Besker T, Martini A, Bosch J. Managing architectural technical debt: a unified model and systematic literature review. *J Syst Softw*. 2018;135:1-6. doi:10.1016/j.jss.2017.09.025
23. Behutiye WN, Rodríguez P, Oivo M, Tosun A. Analyzing the concept of technical debt in the context of agile software development: a systematic literature review. *Inf Softw Technol*. 2017;82:139-158. doi:10.1016/j.infsof.2016.10.004
24. Fernández-Sánchez C, Garbajosa J, Yagüe A, Perez J. Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study. *J Syst Softw*. 2017;124:22-38. doi:10.1016/j.jss.2016.10.018
25. Kitchenham BA, Budgen D, Brereton OP. Using mapping studies as the basis for further research–a participant-observer case study. *Inf Softw Technol*. 2011;53(6):638-651. doi:10.1016/j.infsof.2010.12.011
26. Petersen K, Vakkalanka S, Kuzniarz L. Guidelines for conducting systematic mapping studies in software engineering: an update. *Inf Softw Technol*. 2015;64:1-8. doi:10.1016/j.infsof.2015.03.007
27. Kitchenham B, Charters S. Guidelines for performing systematic literature reviews in software engineering. Tech. Rep. EBSE 2007-001, Keele University and Durham University Joint Report.
28. Dybå T, Dingsøyr T. Empirical studies of agile software development: a systematic review. *Inf Softw Technol*. 2008;50(9-10):833-859. doi:10.1016/j.infsof.2008.01.006
29. Alves NS, Ribeiro LF, Caires V, Mendes TS, Spínola RO. Towards an ontology of terms on technical debt. In 2014 Sixth International Workshop on Managing Technical Debt 2014; 1-7.
30. Caballero Espinosa Eduardo Anel. Understanding social debt in software engineering. PhD thesis. Department of Computer Science in the Graduate School of the University of Alabama; 2021. http://ir.ua.edu/handle/123456789/8278
31. Edmondson AC, McManus SE. Methodological fit in management field research. *Acad Manage Rev*. 2007;32(4):1155-1179. doi:10.5465/amr.2007.26586086
32. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in Software Engineering*. Springer Science & Business Media; 2012. doi:10.1007/978-3-642-29044-2
33. Unterkalmsteiner M, Gorschek T, Islam AM, Cheng CK, Permadi RB, Feldt R. Evaluation and measurement of software process improvement—a systematic literature review. *IEEE Trans Softw Eng*. 2011;38(2):398-424. doi:10.1109/TSE.2011.26
34. Calero C, Piattini M (Eds). *Green in Software Engineering*, Vol. 3. Cham: Springer; 2015. doi:10.1007/978-3-319-08581-4

How to cite this article: Ahmad MO, Gustavsson T. The Pandora's box of social, process, and people debts in software engineering. *J Softw Evol Proc*. 2022;e2516. doi:10.1002/smr.2516

APPENDIX A: DEFINITION OF VARIOUS TYPES OF DEBT

TD type	Definition
Technical debt	"The debt incurred through the speeding up of software project development which results in a number of deficiencies ending up in high maintenance overheads" "A design or construction approach that is expedient in the short term but that creates a technical context in which the same work will cost more to do later than it would cost to do now (including increased cost over time)" "A collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. TD presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability"
Requirements TD	"Refers to the distance between the optimal requirements specification and the actual system implementation, under domain assumptions and constraints"
Architectural TD	"Is caused by architecture decisions that make compromises in some internal quality aspects, such as maintainability"
Design TD	"Refers to technical shortcuts that are taken in detailed design"
Code TD	"Is the poorly written code that violates best coding practices or coding rules. Examples include code duplication and over- complex code"
Test TD	"Refers to shortcuts taken in testing. An example is lack of tests (e.g., unit tests, integration tests, and acceptance tests)"
Build TD	"Refers to flaws in a software system, in its build system, or in its build process that make the build overly complex and difficult"
Documentation TD	"Refers to insufficient, incomplete, or outdated documentation in any aspect of software development. Examples include out-of-date architecture documentation and lack of code comments"
Infrastructure TD	"Refers to a sub-optimal configuration of development-related processes, technologies, supporting tools, etc. Such a sub-optimal configuration negatively affects the team's ability to produce a quality product"
Versioning TD	"Refers to the problems in source code versioning, such as unnecessary code forks"
Defect TD	"Refers to defects, bugs, or failures found in software systems"
Service debt	The need for web service substitution could be driven by business or technical objectives. The substitution can introduce a TD, which needs to be managed, cleared and transformed from liability to value-added. Technical debt can cover several dimensions, which are related to selection, composition, and operation of the service
People debt	Refers to people issues that, if present in the software organization, can delay or hinder some development activities. An example of this kind of debt is expertise concentrated in too few people, as an effect of delayed training and/or hiring
Social debt	Social debt is analogous to technical debt in many ways: It represents the state of software development organizations as the result of "accumulated" decisions. In the case of social debt, decisions are about people and their interactions
Process debt	Refers to inefficient processes, for example, what the process was designed to handle may be no longer appropriate
Sustainability debt	It refers to "the hidden effect of past decisions about software-intensive systems that negatively affect economic, technical, environmental, social, and individual sustainability of the system under design"[15]
Environmental sustainability debt	"the cost (in terms of resource usage) of delivering a software system with a greenability degree under the level of the nonfunctional requirements established by stakeholders, plus the incurring cost required to refactor the system in the future"[34]

Source: Extended version of Lenarduzzi et al.[27]

APPENDIX B: PRIMARY STUDIES

S1. Freire, S., Rios, N., Mendonça, M., Falessi, D., Seaman, C., Izurieta, C., & Spínola, R. O. (2020). Actions and impediments for technical debt prevention: results from a global family of industrial surveys. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (pp. 1548-1555).

S2. Canedo, E. D., Mendes, F., Cerqueira, A., Okimoto, M., Pinto, G., & Bonifacio, R. (2021). Breaking one barrier at a time: how women developers cope in a men-dominated industry. In Brazilian Symposium on Software Engineering (pp. 378-387).

S3. Mandić, V., Taušan, N., & Ramač, R. (2020). The prevalence of the technical debt concept in Serbian it industry: results of a national-wide survey. In Proceedings of the 3rd International Conference on Technical Debt (pp. 77-86).

S4. Martini, A., Besker, T., & Bosch, J. (2020). Process debt: a first exploration. In 2020 27th Asia-Pacific Software Engineering Conference (APSEC) (pp. 316-325). IEEE.

S5. Ramač, R., Mandić, V., Taušan, N., Rios, N., Freire, S., Pérez, B., & Spinola, R. (2022). Prevalence, common causes and effects of technical debt: results from a family of surveys with the IT industry. Journal of Systems and Software, 184, 111114.

S6. Tamburri, D. A. (2019). Software architecture social debt: managing the incommunicability factor. IEEE Transactions on Computational Social Systems, 6(1), 20-37.

S7. Besker, T., Ghanbari, H., Martini, A., & Bosch, J. (2020). The influence of technical debt on software developer morale. Journal of Systems and Software, 167, 110586.

S8. Dreesen, T., Hennel, P., Rosenkranz, C., & Kude, T. (2021). "The second vice is lying, the first is running into debt." Antecedents and mitigating practices of social debt: an exploratory study in distributed software development teams. In Proceedings of the 54th Hawaii International Conference on System Sciences (p. 6826).

S9. Catolino, G., Palomba, F., Tamburri, D. A., Serebrenik, A., & Ferrucci, F. (2020). Refactoring community smells in the wild: the practitioner's field manual. In Proceedings of the acm/ieee 42nd international conference on software engineering: Software engineering in society (pp. 25-34).

S10. Tamburri, D., Kazman, R., & Van den Heuvel, W. J. (2019). Splicing community and software architecture smells in agile teams: an industrial study.

S11. Martini, A., Stray, V., & Moe, N. B. (2019). Technical-, social-and process debt in large-scale agile: an exploratory case-study. In International Conference on Agile Software Development (pp. 112-119). Springer, Cham.

S12. Tamburri, D. A., Kazman, R., & Fahimi, H. (2016). The architect's role in community shepherding. IEEE Software, 33(6), 70-79.

S13. Liebel, G., Tichy, M., Knauss, E., Ljungkrantz, O., & Stieglbauer, G. (2018). Organisation and communication problems in automotive requirements engineering. Requirements Engineering, 23(1), 145-167.

S14. Rios, N., Spínola, R. O., Mendonça, M., & Seaman, C. (2020). The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil. Empirical Software Engineering, 25(5), 3216-3287.

S15. Malakuti, S., & Ostroumov, S. (2020). The quest for introducing technical debt management in a large-scale industrial company. In European Conference on Software Architecture (pp. 296-311). Springer, Cham.

S16. Yli-Huumo, J., Maglyas, A., & Smolander, K. (2014). The sources and approaches to management of technical debt: a case study of two product lines in a middle-size Finnish software company. In International Conference on Product-Focused Software Process Improvement (pp. 93-107). Springer, Cham.

S17. Tamburri, D. A., Kruchten, P., Lago, P., & Vliet, H. V. (2015). Social debt in software engineering: insights from industry. Journal of Internet Services and Applications, 6(1), 1-17.