Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

IT managers' perspective on Technical Debt Management[☆]

Marion Wiese [a,*], Klara Borowa [b]

[a] Universität Hamburg, Department of Informatics, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
[b] Warsaw University of Technology, Institute of Control and Computation Engineering, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland

ARTICLE INFO

Article history:
Received 11 October 2022
Received in revised form 23 January 2023
Accepted 5 April 2023
Available online 10 April 2023

Keywords:
Technical Debt
Technical Debt Management
Technical Debt causes
Technical Debt consequences
Technical Debt communication
Legacy systems

ABSTRACT

Context: Technical Debt (TD) is a term for software solutions that are beneficial in the short-term but impede future change.
Goal: Previous research on TD indicates various management-related causes. We analyze the perspective of IT managers on TD since they usually have a major influence on deadlines, the project's budget, and setting up a TD management (TDM) process.
Method: To determine the IT managers' perspective, we obtained and analyzed data from 16 semi-structured interviews and a three-person focus group discussion.
Results: We found that all IT managers understood the TD concept. They consider TDM to be an essential topic, though nearly none of them had set up a TDM process so far. We identified three major concerns the IT managers had regarding TDM: communicating about TD, establishing a TDM process, and dealing with vintage systems, i.e., old legacy systems We developed a model specifying causes and consequences visible to business stakeholders, causal chains, and vicious cycles.
Conclusions: Our research identifies new research gaps and demonstrates to practitioners that investing in a TDM process may be beneficial. It provides the V4CTD model of *Visibility, Cycles & Chains of Causes & Consequences of TD*, extending the TD conceptual model and facilitating communication on TD with business stakeholders.

Editor's note: Open Science material was validated by the Journal of Systems and Software Open Science Board.

1. Introduction

The term Technical debt (TD) was first used by Cunningham (1992) as a metaphor to explain how delivering software swiftly, results in the need for code rewrites later. The definition of TD has evolved since then. Currently, a single instance of TD, namely a TD item, can be described as a construct that is beneficial in the short term, which hinders further development and maintenance of the software in the long term (Avgeriou et al., 2016; Kruchten et al., 2019). These constructs can occur in various elements of the system or software engineering process, e.g., code, tests, or requirements (Ernst et al., 2021). These days research tends to focus, in particular, on architectural TD (ATD) (Martini and Bosch, 2015; Besker et al., 2018; Verdecchia et al., 2021). ATD was identified as being one of the most dangerous types of TD (Martini and Bosch, 2015), because there is a lack of refactoring strategies that would enable ATD repayment after it is incurred (Besker et al., 2018).

The **rationale** of this study is to focus on the IT managers' perspective on TD since improving the planning and management of TD is a significant factor in minimizing the amount of TD in software projects. In their research on TD causes and consequences, Ramač et al. (2021) show that the largest group of TD causes is related to planning and management. IT managers have a great influence on time and budget resources, i.e., project planning and deadlines, which are the most frequent causes in this category (Ramač et al., 2021). Other researchers likewise found that the effectiveness of Technical Debt Management (TDM) is related to IT managers and their knowledge and overview of TD (Ramasubbu and Kemerer, 2019; Wiese et al., 2022).

Junior et al. identified a lack of research on the business perspective of TD (Junior and Travassos, 2022). IT managers are usually in direct contact with the business stakeholders and negotiate time and budget with them. Therefore, they may have a better understanding of the business managers' perspective on TD.

While the developer's (Rocha et al., 2017), architects (Pérez et al., 2021) and even project manager's (Gomes et al., 2022)

☆ Editor: Christoph Treude.
* Corresponding author.
E-mail addresses: marion.wiese@uni-hamburg.de (M. Wiese), klara.borowa@pw.edu.pl (K. Borowa).

https://doi.org/10.1016/j.jss.2023.111700
0164-1212/© 2023 Elsevier Inc. All rights reserved.

perspectives are taken into consideration by dedicated studies, no study focusing on the IT managers' perspective on TD has been performed so far to the best of the authors' knowledge. Gomes et al. (2022) recent research underlines the significance of this gap by showing that the perspective of management practitioners might be very different from that of software developers.

Therefore, the **goal** of this study is to research the perspective of IT managers on TD by conducting 16 interviews and one three-person focus group discussion. This might provide research gaps or research areas with insufficient focus and enable IT managers to benefit from the analysis of the participant's insights.

Guided by our **research questions (RQ)**, we identified the current knowledge, three main concerns, and potential improvement strategies that IT managers have regarding TD.

RQ 1: What is the current situation regarding TD in the participants' companies?

- ***RQ 1.1.: What and how much do IT managers know about TD?*** To evaluate the involvement of IT managers in TDM, we identified how IT managers interpret the term TD and the knowledge they have about the concept of TD. Misunderstandings in this regard may be a reason for IT managers to not manage TD. Moreover, IT managers may provide previously unidentified knowledge about the concept and mechanics of TD as they are in direct communication with the business managers.
- ***RQ 1.2.: How do IT managers currently manage TD?*** To identify issues with and potential improvements for the current TDM process, we summarized the current state of TDM in the participants' companies.

RQ 2: What are IT managers' main concerns regarding TDM? The IT managers' main concerns regarding TD may differ from the developers' or architects' point of view. The results of this RQ may allow researchers to focus on relevant topics regarding TD.

RQ 3: What measures do IT managers propose to improve TDM? By answering this question, we aimed to identify further potential for TDM improvement. IT managers may be aware of potential improvements to TDM, that are not part of the TD body of knowledge so far.

As the **contributions** of our study, we answered the research questions and gathered the following major insights:

1. The TD metaphor is used as an ambiguous informal term and is not helpful in business communication.
2. There are causes and consequences on different visibility levels for business stakeholders, IT managers, and development teams, which might result in communication issues.
3. We propose to extend the TD conceptual model to include chains and cycles of causes and consequences.
4. To improve (developer) communication and increase the overall awareness of TD, an established TDM process is essential.
5. To overcome the status quo bias, IT managers need more guidance on how to establish a TDM process, preferably through using already established processes and simple methods.
6. Research on how to migrate vintage systems (i.e., systems from the 90s/old legacy systems) and on the possibility of keeping and changing them seems to be still lacking.

With regards to the second and third topics, we extended the conceptual model of TD (Avgeriou et al., 2016) and created the V4CTD model of *Visibility, Cycles & Chains of Causes & Consequences of TD*. This model not only explains the relationships between various causes and consequences but it can also be used to prepare a three-minute pitch to explain a project's TD issues to a business stakeholder.

In the following **subsections**, we present the background information that forms the basis for motivating and understanding this study. In Section 2, we explain our research methodology. We present the results, including the V4CTD model, and answer the research questions in Section 3. Afterward, we discuss our findings with regard to the identified concerns and suggest further research activities in Section 4. We compare our study with related work in Section 5 and discuss the threats to validity of our study in Section 6. With Section 7 we conclude our paper and summarize the contributions for practitioners and researchers.

1.1. Technical debt management

TDM has been intensively researched over the last decade (Li et al., 2015; Rios et al., 2018). In recent years, this resulted in publishing comprehensive management guidelines (Kruchten et al., 2019; Ernst et al., 2021) accessible to software practitioners.

Organizations establishing TDM can achieve various levels of maturity, depending on their current approach to TDM. Yli-Huumo et al. (2016) describe three levels of organizational maturity (no conscious TDM practices, partial implementation of TDM activities, and fully organized TDM). Kruchten et al. (2019) distinguish five levels of TD awareness (from no awareness of the term TD at all to having complete control over TD). Similarly, Martini et al. (2018) describe seven levels of adoption for TD tracking. However, full maturity regarding TDM is not yet a widespread phenomenon in an industrial setting. This can be clearly observed by the results of the survey conducted by Martini et al. (2018), where only 7.2% of respondents reported having a systematic approach to TD tracking.

1.2. TD conceptual models

There is a rich body of work on how technical debt and various concepts related to it should be modeled, called, and described. An overall conceptual model, which has been used as a basis for other research, was presented by Avgeriou et al. (2016). The tertiary studies on TDM by Rios et al. (2018) and by Junior and Travassos (2022) both provide extensions to the conceptual model regarding TD activities, TD types, and strategies & technologies. A similar model, designed to describe architectural debt, has been recently proposed in the work of Verdecchia et al. (2021).

1.3. TD types

Various works tackle the problem of describing possible TD types, depending on what part of the software or aspect of its development the TD item influenced. In their tertiary studies, Rios et al. (2018) found 15 different TD types, while Junior and Travassos (2022) aligned the types with the software development phases. Furthermore, new types like security debt (Ahmadjee et al., 2021) are still emerging. Moreover, TD type names are not always consistent across different works. For example, "People debt" defined Rios et al. (2018) is a very similar concept to "Management and Social Debt" defined by Ernst et al. (2021).

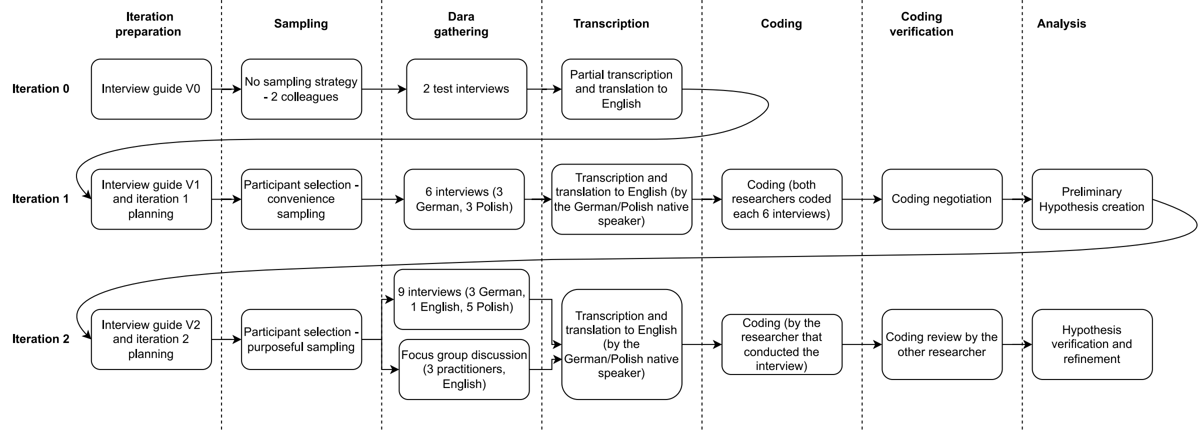
Iteration preparation
Sampling
Dara gathering
Transcription
Coding
Coding verification
Analysis
Iteration 0
Interview guide V0
No sampling strategy - 2 colleagues
2 test interviews
Partial transcription and translation to English
Iteration 1
Interview guide V1 and iteration 1 planning
Participant selection - convenience sampling
6 interviews (3 German, 3 Polish)
Transcription and translation to English (by the German/Polish native speaker)
Coding (both researchers coded each 6 interviews)
Coding negotiation
Preliminary Hypothesis creation
Iteration 2
Interview guide V2 and iteration 2 planning
Participant selection - purposeful sampling
9 interviews (3 German, 1 English, 5 Polish)
Focus group discussion (3 practitioners, English)
Transcription and translation to English (by the German/Polish native speaker)
Coding (by the researcher that conducted the interview)
Coding review by the other researcher
Hypothesis verification and refinement

Fig. 1. Study design.

1.4. Technical debt causes and consequences

The main causes and consequences of TD have been extensively described in previous research, particularly by the InsighTD project (Anon, 2022a; Rios et al., 2020; Ramač et al., 2021), where a unidirectional model of TD causes and consequences was presented.

Notably, causes associated with management, dominate many lists of TD causes. Rios et al. (2019) found that in 34% of the TD occurrences, TD is incurred as a result of a cause from the "Planning and management" category. Ramač et al. (2021) carried out an extensive survey that found 725 "Planning and Management" cause occurrences, with the second most populated category being "Development Issues" (473 occurrences). Vogel-Heuser and Bi (2021) identified that 32% of their researched TD items were initiated by management decisions.

Regarding the consequences, Kruchten et al. expanded the unidirectional model by dividing the effects into consequences (direct results of the TD item) and symptoms (the visible impact of the TD item) (Kruchten et al., 2019). This idea is rooted in works on TD conceptual models (Avgeriou et al., 2016; Verdecchia et al., 2021). However, Stochel et al. (2020) identified that a *"smell may be considered as a symptom of a TD Item, rather than the TD Item itself"*., which contradicts this idea of visibility, at least on a business level.

1.5. Managers' perspective on TDM

A particularly striking detail from the work of Rios et al. (2020) is that participants openly stated that their project managers are ineffective when it comes to managing TD. One participant even bluntly states that project managers (and customers) simply rush to "receive something working as soon as possible" (Rios et al., 2020). Additionally, Soliman et al. (2021) show how communication between the development team and management greatly impacts debt incurring architectural design decisions. What is more, in their case study, Wiese et al. (2022) describe the impact a TDM process may have on IT managers in terms of TD overview, managing the project pipeline, and communication with their business stakeholders.

2. Method

The aim of our study was an in-depth exploratory investigation of the IT managers' perspective on TD based on their real-world experiences. Thus, we decided on an inductive (Runeson et al., 2012) approach for the study design, which follows an "iterative observational research path" (Stol and Fitzgerald, 2015). Additionally, in order to heighten the relevance of our findings (Jain et al., 2013), we flexibly adjusted each of the three iterations of the study. The overall study design is presented in Fig. 1.

Each of the three iterations had a different goal:

- **Iteration 0** served as a test of our data gathering method. We tested the interview guide, discussed and adjusted the overall study design by conducting two test interviews. The data from these interviews was not used for further analysis.
- **Iteration 1**, during which we performed 6 interviews in total, was supposed to enable the creation of preliminary findings and the creation of a stable code list that would be used for further analysis.
- **Iteration 2** was supposed to provide enough data to either prove or disapprove the preliminary findings and allow us to refine our findings.

We concluded the study after these iterations since we achieved code saturation (Saunders et al., 2018) as described in Section 2.1.

The study iterations (with the exception of the simplified Iteration 0) consisted of the following six steps, which were performed slightly differently to meet each iteration's goal:

(1) Iteration preparation: We discussed the results from the previous iteration and adjusted the plan for the upcoming one. During the preparation step, the authors decided on the sampling strategy, revised the interview guide, and planned the coding and analysis strategy. *(2) Sampling:* We recruited participants for our study, depending on the sampling strategy chosen for the iteration (see Section 2.1). The procedure of gathering data through interviews is described in Section 2.2.1, and the focus group discussion is explained in detail in Section 2.2.2. *(3) Transcription:* Since data was gathered in 3 languages (English, German and Polish), the transcriptions of the interviews were translated to English. The translation process was always performed by the German or Polish native-speaking author to avoid any misunderstandings. *(4) Coding:* We coded the transcripts according to the coding process explained in Section 2.3. *(5) Coding verification:* We verified and refined the coding as presented in Section 2.3. *(6) Analysis:* We discussed the results of the coding and memoing. More details are given in Section 2.3.

2.1. Sample

Before gathering participants, we first defined a set of roles to better specify the targeted participants and differentiate them from other roles:

Table 1
Study participants.

Participant	Company domain	Company size (no. of employees)	Team size (no. of employees)	Years working in IT	Years as an IT manager	Gender	Top manager	Former manager
I1	Finances	100–1000	25–100	>10	5–10	m	x	
I2	Hardware & Software Engineering	1000–5000	0–10	>10	2–5	m		
I3	Online Retail Services	1000–5000	0–10	>10	>10	m		
I4	Software Engineering	0–100	0–10	>10	>10	m	x	
I5	Media	1000–5000	>100	>10	>10	m	x	
I6	Human Resource Services	0–100	10–25	>10	>10	m	x	
I7	Food industry	> 5000	0–10	>10	>10	m		
I8	Software Engineering	0–100	0–10	>10	5–10	m	x	
I9	Finance	1000–5000	25–100	>10	>10	m		
I10	Logistics + Finance	100–1000	25–100	>10	>10	m	x	
I11	Telecommunications	> 5000	>100	>10	2–5	f		
I12	Human Resource Services	0–100	0–10	>10	0–2	m		
I13	Fast Moving Consumer Goods	> 5000	0–10	5–10	0–2	f		
I14	Entertainment	1000–5000	10–25	5–10	2–5	f		x
I15	Online Media	100–1000	0–10	>10	5–10	m		x
I16	Entertainment	1000–5000	10–25	2–5	0–2	m		
F1	Social Media	1000–5000	0–10	>10	5–10	f		
F2	Social Media	1000–5000	0–10	>10	2–5	m		
F3	Social Media	1000–5000	0–10	0–2	0–2	m		

Table 2
New codes with each transcript.

Interview No.	I1	I2	I3	I4	I5	I6	Focus group	I7	I8	I9	I10	I11	I12	I13	I14	I15	I16
No. of codes	52	26	22	25	30	46	16	38	8	16	18	22	9	13	8	4	2

IT managers have personnel responsibility for their development teams, i.e., they are IT line managers, and can be on different levels of the company's hierarchy. *Top managers* are IT managers that are either their companies' CTOs or are more than one hierarchy level above the developers.

As the *development team or developers*, we summarize all technical roles that do not have personnel responsibilities but solely serve the software development process, e.g., developers, project managers, product owners, and testers.

Business stakeholders can be business managers or customers. They both have the same roles regarding TDM and are only called differently depending on whether they are a part of the developing company or not.

As suggested by Baltes and Ralph (2022), we defined the goal of our sampling strategy before starting data gathering. We decided that our goal was achieving code saturation (Saunders et al., 2018), which is the standard way to assess that further data collection is not necessary.

To answer our questions appropriately, we sought IT managers that had to be responsible for scheduling projects, or at least tickets, or they had to directly communicate with the business stakeholders. This resulted in 19 IT managers; 14 of them had all required properties, eleven had additional budget responsibilities, and six were top managers. The IT managers worked in various domains, as shown in Table 1. Seven interviewees worked for German and nine for Polish companies. All three focus group participants were employed in a single German company.

Since iteration 0 was not supposed to be used for data gathering, we simply performed the test interviews with two practitioners experienced in software development.

During iteration 1, we gathered participants through convenience sampling (Baltes and Ralph, 2022), i.e., through our personal networks. Since the iteration's purpose was to create preliminary hypotheses and establish a common coding approach, we gathered participants from very different domains in order to obtain a diverse set of data. At this point, we did not take other participant characteristics into account.

For iteration 2, we decided on purposeful sampling (Baltes and Ralph, 2022) to find all possible codes, which would allow us to get to the saturation point. Having analyzed the data from iteration 1, we made observations about the types of participants we were missing, such as participants with less than five years of experience, from a creative domain, from a company with more than 5000 employees, ex-managers, and female managers. We purposefully sought participants that would meet these criteria for this iteration.

We counted the appearances of new codes after coding each transcript (see Table 2). During the coding of the last transcript, only 2 codes (of 393) emerged, which is under 0.5% of all codes. Thus, we concluded that saturation was achieved.

All participants filled out a consent form and a questionnaire about themselves (Wiese and Borowa, 2022) before participating. An overview of all participants is presented in Table 1.

2.2. Data collection

We collected our data through leading sixteen interviews and one three-person focus group discussion that focused on our preliminary findings. In the following subsections, we give detailed information on these two data gathering methods.

2.2.1. Interviews

Table 3 shows an overview of all interview questions (IQ) and their relation to the RQs. The interviews were performed in the form of semi-structured interviews, which means that slight variations from the overall procedure (like asking follow-up questions) were allowed.

Since we found that some key topics were missing in iteration 1, some of the questions were added during iteration 2 to maximize the relevance of our findings (Jain et al., 2013), Those additional questions and the rationale behind them are:

Question 1.a.: Participants did not use the financial metaphor , but instead often used their own metaphors. We wanted to explore the metaphor's usefulness.

Question 6: IT managers mostly talked about the existing TD and not much about incurring TD intentionally. We wanted to explore incurring TD intentionally further.

Question 7. and 7.a: IT managers did not explicitly mention prevention mechanisms, which we wanted to explore as well.

Table 3
Interview questions.

IQ No.	Question	Related RQ
1.	What is TD to you?	RQ1
1.a	Does the metaphor help to understand the concept?	RQ1
2.	How does your team/group manage TD?	RQ3
2.a.	Do you have an overview of your team's TD?	RQ2, RQ3
3.	What do you do to help manage/enable the management of TD?	RQ3
3.a	What is your part/role in managing TD?	RQ4
4.	What TD items does your team/group have - could you give some examples?	RQ1
5.	From these TD items, which one do you consider the most impactful?	RQ1, RQ2
5.a	How/Why did this [item] happen?	RQ2
5.b	What influence/consequence does this TD item have? (What is the problem with this TD item?)	RQ1
5.c	Did you have any influence for it to be incurred?	RQ1, RQ3
5.d	Do/Did you have any influence for it to be removed?	RQ1, RQ3
5.e	What could have been done better when dealing with this TD item?	RQ 3
6.	At which point in time do you get to know of TD? (While incurring them or later?)	RQ1, RQ2
6.a	What would be the advantages (or disadvantages) of early recognition?	RQ1, RQ2
7.	Do you think TD can be prevented?	RQ2, RQ3
7.a	Do you know of TD items that could have been prevented? (Which ones? What are the preconditions for this?)	RQ2, RQ3
8.	From your perspective, is it important and worthwhile to manage TD? And why?	RQ2
9.	How could you, as a manager, benefit from TD management?	RQ2
10.	In your opinion, as a manager, what are the drawbacks of TD management?	RQ2
11.	What do you think you could do (optimize) to manage TD (better)?	RQ3
11.a	What do you need from your developers to better manage TD?	RQ3
11.b	What do you need from your upper management or customer to better manage TD?	RQ3
12.	For former managers:	
12.a	When have you stopped being a manager?	
12.b	Why did you decide to change your role?	
13.	Is there anything you would like to add?	

Questions 11.a and 11.b: IT managers often blamed their environment for the lack of an effective TDM process and a growing amount of TD items. To identify TDM's potential improvements, we asked them about their preferred environment for better TDM.

Question 12: To overcome the survivorship bias, we included former managers, who have resigned from their positions, during iteration 2. Question 12 was used to obtain additional information about them.

2.2.2. Focus group discussion

We used the focus group discussion to provide method triangulation (Runeson et al., 2012), which reduces the bias of gathering data through only one method. To perform method triangulation, data has to be gathered through different methods. In the case of the focus group discussion, we presented our preliminary hypotheses, which were the product of the iteration 1 analysis, to a group of three IT managers from one company. During the discussion, we asked them to agree or disagree with the findings and to discuss them with each other. The following preliminary findings were covered:

- The IT managers are concerned about software aging. This topic also comprised the problems regarding iterative TD repayment and the impact this topic has on old-established companies,
- The IT management evolved over the last decades and the possible next step would be to adapt TDM. This concern included TDM issues in agile environments and the need for a responsible person to keep track of TD.
- Communication is a major source of problems regarding TDM with reference to all stakeholders. We discussed the communication with business stakeholders, including the suitability of the metaphor and causes for the business stakeholders' "it works" mentality.
- The mindsets regarding decision-making need to change to avoid TD, and important decisions must be: (1) recognized as such, (2) made consciously, (3) made by groups, (4) based on a pragmatic evaluation through a comparison of options.

The slides used for the focus group discussion are available as part of the additional material (Wiese and Borowa, 2022).

2.3. Data analysis

We used "The Coding Manual" from Saldaña (2013) for orientation on the optimal coding procedure and the MAXQDA tool (Anon, 2022b) to perform the coding.

The coding techniques that we used (all of them originally described in Saldaña (2013)) included:

- Structural Coding: marking a segment of the transcript as one containing data about a certain topic. For example, we used the code "TD knowledge - causes" to mark segments where participants mentioned specific information about any TD causes. The Structural Codes are the only codes that were planned before the analysis started. All of them were related to the RQs.
- Process Coding: Process coding was used to mark segments of the transcripts that were related to specific actions associated with TDM. All process codes contained a verb in the gerund form ("ing"). For example, "Incurring TD" was used to mark the part of the interview when IT managers talked about the specifics of how a TD item was incurred.
- Value Coding: Value codes were used to mark participants' values, attitudes, and beliefs about TD and TDM. These codes had names that were always preceded by "V:" for Value, "A:" for Attitude, or "B:" for Belief. For example, "V: TD(M) is important topic" was a code used to mark a segment where the participant stated that they believe that the topic of managing TD is of importance to their work.
- Descriptive Coding: Using codes that summarize the data from a particular segment. For example, the code "Good Developers are leaving/harder to hire" was used to mark the part of the transcripts where participants' mentioned that developers from their teams left their company because of TD, and it may be hard to hire new employees thereafter.

During the coding process, we constantly reviewed, extended, and reorganized the coding list. Codes were usually organized in a structure of sub-codes, with the structural codes as the main parent codes. For example, in "Structural \TD knowledge - consequences \Business consequences \visible - Business problems \Losing the customer's trust" the code "Losing the customer's trust" is used to mark data segments where participants mention how they lost the customer's trust - which was a business consequence visible to the business stakeholders.

Additionally, at any point in the study, we created analytic memos, which are all available in the additional material (Wiese and Borowa, 2022). Memos are notes related to particular parts of the data (like quotes from the transcript, codes, groups of codes, or notes from the interviews) that contain various observations from the researcher performing the analysis. They are often a source for working hypotheses that can be explored later during the study. For example, the memo titled ”The TD metaphor is not used/filled out“ was related to the observation that the TD metaphor may not be widely used and understood.

For **iteration 1** coding, we used all of the aforementioned coding methods.

We used the coding guidelines of Campbell et al. (2013) to establish a common coding scheme, which allows a single coder to perform the analysis in the next iteration.

We solved the ”Unitization Problem With Units of Meaning“ (Campbell et al., 2013) (problems with the mismatch of the length of the coded text) by deciding that the segments that we will be coding should always be blocks between punctuation marks.

We applied the negotiated agreement approach (Garrison et al., 2006) to resolve the ”Discriminant Capability Problem“ (Campbell et al., 2013) (discrepancies in coding by different coders). This means that all six iteration 1 interviews were coded by both authors separately. Subsequently, all of the differences in the coding were discussed until a full consensus was reached. The organization of the sub-codes was an additional crucial part of these negotiations.

Furthermore, the iteration 1 coding and memoing allowed us to establish a set of preliminary findings and hypotheses, which we focused on during iteration 2.

For **iteration 2**, the interviews were coded by the researcher who led the interview to avoid misunderstandings. To reduce researcher bias, the second author reviewed the coded transcripts. During this stage, we, particularly, gathered data that filled the gaps in our coding scheme, which enabled us to fully develop our hypotheses and reach code saturation.

When exploring the causes and consequences of TD items, we used the conditional/consequential matrix described by Corbin and Strauss (2014), which is a method used to analyze the interplay between a process and its conditions/consequences on various levels (from the level of an individual person, through a group they are part of, to an international one). This allowed us to view the causes and consequences of TD incurrence in the context of their visibility for different stakeholders.

3. Results

In this section, we present the results of our research questions. We give a detailed overview of the IT managers' existing knowledge about TD in Section 3.1, present the IT managers' main concerns in Section 3.3, and summarize the potential improvements as perceived by the IT managers in Section 3.4.

3.1. RQ1.1: Managers' knowledge about technical debt

To answer our first question about the IT managers' current knowledge of TD, we summarize their knowledge regarding the Definition of the TD term and their perception of the metaphor in Sections 3.1.1 and 3.1.2. After that, we identify the mentioned TD types in Section 3.1.3 and give an overview of the IT managers' team's maturity/awareness levels in Section 3.2. We continue by identifying the elements of the TD conceptual model (Avgeriou et al., 2016) that IT managers are aware of, and the relationships between causes and consequences they describe in their examples in Section 3.1.4. In Section 3.1.5, we introduce the concept of TD's visibility for different stakeholders. Finally, we give a detailed description of the identified TD cause and TD consequence categories in Sections 3.1.6 and 3.1.7.

3.1.1. Definition

Most IT managers (17 out of 19), were familiar with the concept of TD. One participant believed it to be a colloquialism with no specific definition, and only one IT manager had never heard the term before. IT managers used the following key concepts to define the term TD (number of IT managers in brackets):

- TD consequences (15), e.g., *"[It] means that systems … are not flexible enough to be further developed, iterated on, added new features or additional functionality to"*.
- TD causes (12), e.g., *"[I]it can happen that we - but this happens only rarely - we purposely develop something faster from the time perspective"*.
- TD short-term benefits versus long-term consequences (9), e.g., *" And then we discussed various options and there was always a fast track. But maybe that [fast] solution is not very good maintainable"*.
- Process descriptions (incurring, repaying) (4), e.g., *"[F]or me it means, in general, code that we want to refactor…"*.
- TD types (4), e.g., *"[S]uch a debt can manifest itself in different places, be it at the level of the application's source code or at the level of some architectural mistakes"*.
- TD as old or obsolete technology (3), e.g., *"So by technical debt I mean something …, which from an IT perspective is obsolete technology in the sense of: I don't have know-how anymore"*.
- Financial metaphor (1), e.g., *"Technical debt, that's kind of an analogy, a little bit to debt that you take on in the financial world"*.

3.1.2. Metaphor

Unprompted, only one IT manager mentioned the financial metaphor while defining the term TD. However, IT managers often used other metaphors when explaining the TD concepts:

- A progressing knee disease: TD was compared to a disease that initially does not cause serious problems but, if left untreated, may result in severe consequences such as the need for limb amputation (abandoning the project altogether) i.e., *"[Y]ou can, of course, have your knee hurt all the time and you don't go to the doctor because you think all the time, it's just a little thing, ... And one year after the other goes by, and at some point, your hip hurts because you have a bad posture . . .. And at some point, you go to the doctor and he says: Yes, the only thing that helps is a prosthesis"*.
- Tailoring bespoke clothing: The piece of clothing was compared to a system, and various changes made according to the client's demands (incurring TD was a byproduct of these changes) gradually caused the garment to deteriorate i.e., *"When we plan a software, a digitalization project, then we tailor a piece of clothing. And at the end, when the garment is finished, then comes real life. And then . . . we have to leave it out at the hem, then we have to knit cuffs on it, then he [the client] wants to have another pocket. And after a while, it turns into not such a nice garment anymore"*.
- Building cathedrals: A system was compared to a cathedral. After the construction of a cathedral, the workers responsible for it disappear with the knowledge about how the cathedral was built. This is a metaphor of Documentation TD and knowledge vaporization, i.e., *"Software and cathedrals are much the same: First, we build them, then we pray. That actually sums up the typical project process quite well. . .. And basically, it is like if you stay in the picture, the cathedral builder of the Cologne Cathedral, who ultimately only tackles it when something starts to crumble off somewhere"*.

- Not fitting puzzle pieces: This describes an instance of TD where various system components were developed independently, which caused a problem during an integration process, i.e., *"I can think of more like geometric metaphors, . . . that somewhere there is a mismatch, which is based on the fact that the elements do not fit together"*.
- A freight train: The train was compared to a system that required constant maintenance to run properly. One manager explained this in detail after their interview. This IT manager negotiated TD repayment with a business stakeholder by comparing this to the maintenance costs of a freight train. He was asked by the business stakeholder: *"But the IT doesn't break, why do you need maintenance?"* The IT manager explained: *"We do even worse! We open the engine and operate directly – hands-on – in the engine, and we do this without an operations manual, which you usually have in detail for freight trains"*.
- House building: A house was equaled to an IT system that must be built in a specific way, according to engineering best practices. If a load-bearing wall is not built (incurring TD by making the process faster and cheaper), then a door cannot be mounted (TD causes development to stop) i.e., *"What you can explain to them is, if I don't install a door in a house, then I just can't get in . . .. If I want to have the door somewhere else, well, then I'll just come and mill it in . . .. [N]ow imagine that you've put a load-bearing wall there. The load-bearing wall has to go somewhere. . .. [T]he colleagues from the business are – partly at least – not technically versed, they don't think of something like a T-beam."*.
- Quicksands: A spiral of incurring more and more TD was compared to the deathly danger of being devoured by quicksand, i.e., *"So if we see this technical debt, we notice earlier that what we're doing is only going to get worse, worse, worse, then maybe we can say at an earlier stage - So let's just redesign the feature. Let's just make it from scratch, instead of wading further into these quicksands"*.

Considering this wide variety of metaphors, we additionally asked about the usefulness of the term TD as a metaphor during the second iteration of data gathering (see Section 2.2.1). Most IT managers interpreted the metaphor's or its usefulness in a **negative way**. They stated that *"this analogy points to something negative"* and is not well understood, particularly by business stakeholders. One IT manager compares it to having *" a corpse in the closet"*. Regarding the communication with business stakeholders, one IT manager explains that *"most people in the business can't really imagine what TD is, even though the metaphor itself is actually very apt . .. I think if that were better understood, then we wouldn't have the problem at all"*.

Fewer saw the metaphor in a **positive light** and mentioned that the metaphor helps to explain or understand the TD concept, and recalls the growing interest. One IT manager stated that *"the metaphor with the debt and with the interest and so on is really useful for explanations"*.

Some explained the metaphor is already **widely used in the industry** and therefore a good term, e.g., *"[F]or me it's part of the language of IT specialists or project managers, so for me it's obvious"*. One IT manager notes that *"this term has become established. And it is really used at every corner"*.

3.1.3. Technical debt types

During the interviews, we asked the IT managers for two to three examples of TD items, and then to choose the most impactful one for detailed questioning. We summarized the number of examples and mentions of various TD types and the number of IT managers mentioning them to get an idea, of which types of TD items are most relevant for IT managers.

We noticed, that IT managers frequently mentioned problems regarding the optimal point in time for an update, e.g., *"You then get into a situation where you're collecting a technical debt because you missed the point in time where you can just upgrade things easily"*. Even the infrastructure's version can become a problem, as pointed out by one IT manager: *"And recently we had such problems that we ran out of spare parts, for example, because part of the software was based on computer drivers, which simply could not be obtained anymore"*. Ernst et al. (2021) describe a similar concept on the implementation (code) level but not on an architecture level, contrary to most IT managers. As we could not find a related type on architecture level in literature, we identified **Updating TD** as a specific type of ATD that occurs when third-party components or systems are not updated on time.

Overall, we identified the following occurrences of TD types (number of examples/number of IT managers) :

- Architectural TD (55/6), e.g., if *"a lot of logic was transferred to the database level and, at some point, it turned out that there was quite a lot of traffic in the system and it generally started to lag this system"*.
- Updating TD (21/9), e.g., one IT manager's team was *"more or less surprised by the fact that the support for this component was virtually discontinued by the 3rd party and we just had to react.*
- Code TD (22/10), e.g., *"when you hire new people, young people, for example, where they . . . write some code fragment that is not optimal"*.
- Documentation TD (14/6), e.g., *"somehow they forgot to create . . . proper and orderly documentation, which in the end leads to the fact that there is exactly one developer who has an overview of the entire system"*.
- Requirement TD (13/6), e.g., one IT manager explained regarding a tool that could only deal with one-page forms: *"It was basically, well short-sightedness in terms of the direction of the requirements changes, because you could easily predict that the forms . . . can be multi-page"*.
- Infrastructure TD (10/7), e.g., *" VMWare version 5, still used Flash . . . for the web client. Adobe has ultimately replaced or discontinued Flash and, thus, the accessibility of such VSphere stuff is quite difficult, because you first have to find an old system where Flash Player is still on it"*.
- Test TD (5/4), e.g., was mentioned a few times by the IT managers when explaining what TD is, but no concrete examples were given.
- Security TD (1/1) is mentioned by one IT manager: *"In this case, such situations may even end catastrophically – some leaks of personal data. They can be fatal for very large companies"*.
- Deployment TD (1/1) is mentioned by one IT manager regarding a distributed system: *"[T]his is the process of introducing changes, which requires issuing many orders to many different organizations in order to transfer the change to the production environment"*.

This shows that IT managers are aware of the variety of TD types and the fact that TD is not just a topic for developers related to code.

Among the most impactful items were eleven ATD items, two updating debt items, two requirement debt items, and one code debt item. This means that an overwhelming majority of TD items chosen by the participants were examples of ATD.

3.1.4. Technical debt conceptual model

Comparing our study's results with the conceptual model of TD as proposed by Avgeriou et al. (2016), we identified information about TD items, types, causes, and consequences (incl.

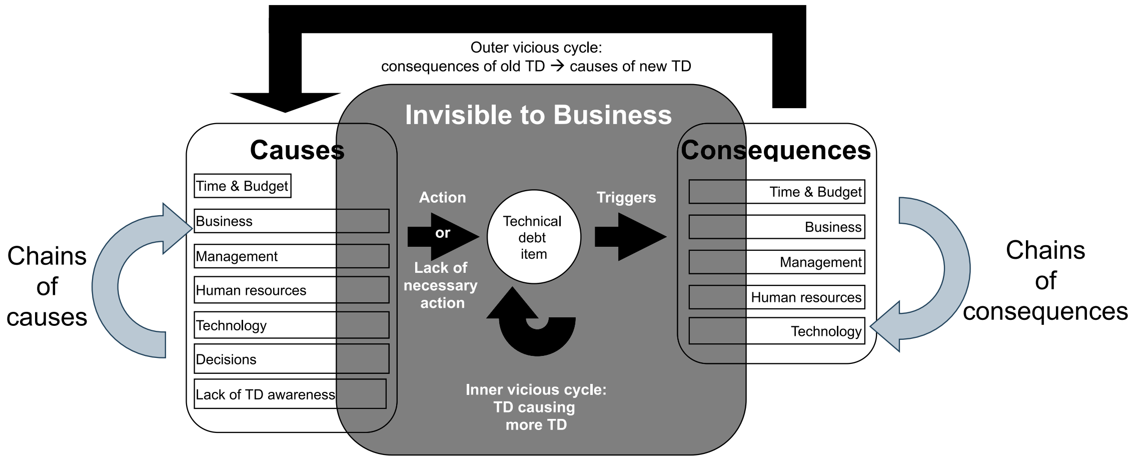

Fig. 2. V4CTD model of *Visibility, Cycles & Chains of Causes & Consequences of TD*.
(gray box shows concepts invisible to business stakeholders, i.e., TD items and part of causes/consequences).

symptoms and business goal influences from Avgeriou et al., 2016) in the transcripts. However, the idea of a unidirectional concept of TD causes leading to a TD item and the TD item leading to TD consequences and symptoms, as in the works presented in Section 1.2, did not perfectly fit our study's results.

We, therefore, suggest extending the conceptual model of TD, with regards to the causes and consequences, as summarized in our **V4CTD model** of *Visibility, Cycles & Chains of Causes & Consequences of TD* in Fig. 2. Below we explain the additions and provide one detailed example for each of them.

Chains of Causes. Various TD causes can trigger other causes. When explaining why a specific TD item was incurred, one manager explicitly stated: *"The problem, as usual, I think there are many layers, reasons"*.

In the following example, time pressure is one cause, that leads to outdated documentation, which is a (secondary) cause for knowledge vaporization. *"Certainly, there was documentation at the beginning, but then it didn't continue for various reasons. And the usual reasons are always time pressure. People leave the company, new people come in. Something serious has been changed in the architecture. No one really understood how to document it, and so on"*.

Chains of Consequences. Various TD consequences can trigger other consequences. This concept is similar to the one of "consequences and symptoms", as described by Avgeriou et al. (2016) and Kruchten et al. (2019), where invisible consequences lead to visible symptoms. However, the V4CTD models' *chain of consequences* includes the possibility of a visible symptom triggering an invisible consequence, e.g., losing a customer may cause stress and decreased morale in the development team.

A chain of consequences where technical consequences lead to business consequences is reported by one IT manager: *"But if you've started to build a complex product internally, where you've got bottlenecks everywhere in the development, then you're not only hindering your software development and driving up costs and time, but you're also actively hindering the further development of the business"*.

Inner Vicious Cycles. The TD items themselves may be the cause of incurring additional TD. This concept was already mentioned as TD vicious cycles, e.g., by Ramač et al. (2021), but evidence or detailed information for this is lacking. However, our participants described such situations: *"So they [the current staff] learned and they already knew, but when new people were introduced into the project, . . . they could get lost [in the code], like: 'Hey, what should I do here, because I don't have this option to choose from?'"* In this case, the missing documentation, which is one TD item, leads to unreadable code, which is another TD item.

Outer Vicious Cycles. The consequences of one TD item may become a cause for incurring more TD. Martini and Bosch (2015) previously described this as a TD vicious cycle. They reported the cycles of specific TD items but did not conceptualize their insights. Similarly, our participants identified cycles where TD consequences led to other TD causes and created an outer vicious cycle, e.g.: " *[W]e would need to redo everything from scratch to make it [a functional change] the proper way. And then there comes a discussion like: 'Do we even have the capacity right now and time in order to implement this stuff?' And in some bad cases, it ends up with another quick and dirty thing on top of already existing, working dirty [things]"* In this case, the consequence of one TD item is the need for a rewrite before another change. Avoiding this rewrite is then the cause for more TD to be incurred.

Moreover, by combining the different descriptions of TD items provided by IT managers, we uncovered that similar issues were often mentioned as either TD causes or TD consequences. For example, dissatisfied and stressed developers leaving the company is a TD consequence. However, a lack of developers can be the cause of TD. Likewise, a consequence on the management level is projects becoming unmanageable when TD has to be repaid unplanned. Bad project management, however, is also a cause for TD. Similarly, time and budget were frequently named as cause and as consequence. On a more abstract level, business stakeholders might cause TD, e.g., by setting tight deadlines. In subsequent iterations, not all required features may be implemented, or the next deadline may not be met due to the interest that has to be paid for that former TD incurrence.

3.1.5. Visibility of causes and consequences

Another result of the study, which has been incorporated into the V4CTD model, is the observation that not every stakeholder may have an in-depth insight into the specific mechanics of every TD item.

IT managers cannot be aware of every detail of the system's development, i.e., they may not know about or understand every TD item that is incurred and the causes and consequences related to this item. Some managers pointed this out when being asked about their influence on the incurrence of their example TD item. For one of the former managers, this had been one of the reasons to quit his position: " *[T]he longer you are in that [IT manager] position, the further away you are from the development and from the awareness that, for example, technical debts have to be reduced continuously"*. This lack of visibility may be the cause

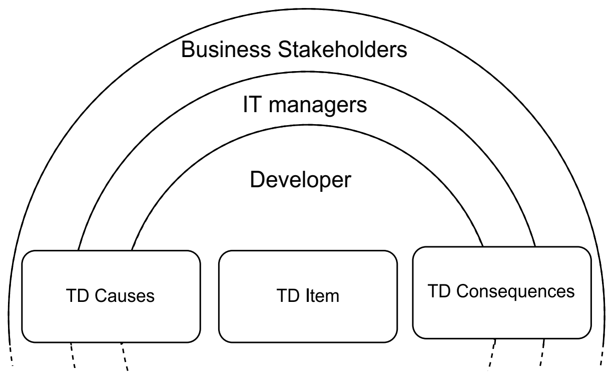

Fig. 3. Visibility and influence cycles of stakeholders.

of faulty decisions, as an IT manager explained: *"And as I said, the conditions were not clear on the table [, i.e., openly available/visible to him]. . .. And that's why you suddenly decide differently than you perhaps should have decided, retrospectively, from today's point of view".*

Analogous, this is, even more, the case for **business stakeholders**, since their understanding of the technical subject is limited and they are less involved in the technical decisions. One top manager pointed out: *"They [business stakeholders] are not professionals. [They say:] 'Why? I bought the software; it still works. That the environment changes continuously with each update, it is not at all clear to them"*. However, business stakeholders are well versed in the field of business goals, which are impacted by the consequences as described in the conceptual model of Avgeriou et al. (2016). Additionally, they also influence the causes, i.e., their decisions may be the cause for a cause as a part of a chain of causes.

In turn, for the **developers**, the rationale for the business stakeholders' or IT managers' debt-incurring decisions may not be visible anymore.

We used the conditional matrix (Corbin and Strauss, 2014) describing activity circles of different stakeholders around an action as an inspiration to structure the causes and consequences into visibility circles for each stakeholder, as summarized in Fig. 3.

- Developers may be aware of the TD item itself and its direct causes and consequences, but they may have limited insights into debt-incurring business decisions.
- Managers may be able to understand the causes and consequences of TD, but may not understand every TD item in detail. Additionally, they have a deeper insight into business goals, and, as such, may see the causes behind causes or the consequences behind consequences.
- Business stakeholders may not understand the technical side of TD, but they can understand and influence some of the causes and suffer from some of the consequences.

In our V4CTD model, we grouped TD causes and consequences into visible and not visible causes and consequences with regard to business stakeholders. In the case of **visible causes**, the root cause is a decision made by business stakeholders, or they may be able to understand the cause. For **visible consequences**, business stakeholders may observe the consequence because the business is directly impacted. The summary of causes, consequences, and their visibility is shown in Fig. 4 and described in detail in the following sections.

3.1.6. Technical debt causes

Through our analysis, we identified the following eight categories of TD causes (number of mentions in brackets): [Faulty] decisions (138), Management causes (60), Business causes (59), Technological causes (43), Limited time (40), Human resource causes (31), Limited budget (16), and Lack of TD Awareness (8). Below, we explain each category and give one example of visible and invisible causes for each category, respectively. An overview is given in Fig. 4 and more details can be found in the additional material (Wiese and Borowa, 2022).

[Faulty] decisions. These decisions include both conscious rational ones, as well as mistakes and sub-optimal or even reckless ones. We observed that decisions were made by different stakeholders:

- Developers make decisions and possibly make mistakes or write unreadable code when implementing algorithms or functionalities.
- IT managers might, for instance, decide on a specific team structure, postpone decisions, or make mistakes during requirements gathering.
- Architectural decision-making may lead to TD, e.g., when not all options were compared, options were missing in former times, or knowledge about an option is lacking.
- A supplier might, for example, decide to discontinue the support for a component, which creates the need to change the running system.
- Business managers, e.g., might decide to set a certain deadline without considering the actual time and effort required or make inconsiderate architectural decisions.

(**Visible:**) Business managers making technological decisions without consulting the IT department are shown in this example: (**Invisible:**) IT managers often mentioned that they did not supervise the developers effectively and did not establish efficient peer reviews: *"[T]his person was getting feedback that he had to write better [code], but it took a long time [until this happened]. So I would say that it was my mistake as a manager that I left that person alone".*

Management causes. These are causes related to IT management, i.e., mostly social or organizational issues that lead to incurring or increasing TD unnecessarily. (**Visible:**) This category includes missing processes in earlier times, i.e., the evolution of IT management: *"So in IT, certain best practices have become established . . . in managing products, in developing software, in managing infrastructure. There are entire process libraries".* (**Invisible:**) Miscommunication issues, e.g., between business and IT stakeholders, is another cause belonging to this category, like *"The developer says: 'Of course, I can develop this for 50 days, – comma if – . . . if I then still have time to take care of refactorings and documentation.' But he never uttered that [comma if] sentence. Nobody ever understood that / heard that, and then nobody scheduled it, and it didn't happen".*

Business causes. These causes are directly or indirectly related to business decisions or knowledge. One commonly known business cause is the need to reduce the time to market and pre-set deadlines by the customer. (**Visible:**) Causes in this category also include the change in business strategy, or a complex and growing business model, e.g., in a start-up situation. One manager mentioned with regards to this category: *"At some point, it reached its limits when we became international, with all the different languages, formatting of numbers, currencies and so on".* (**Invisible:**) A more indirect cause is the lack of knowledge of the requirements, e.g.: *"We started preparing a project with that, like this. And we realized just in time that we were in the process of building up technical debt again because then the [new] idea came up that you could also use this smart card to track work processes".*

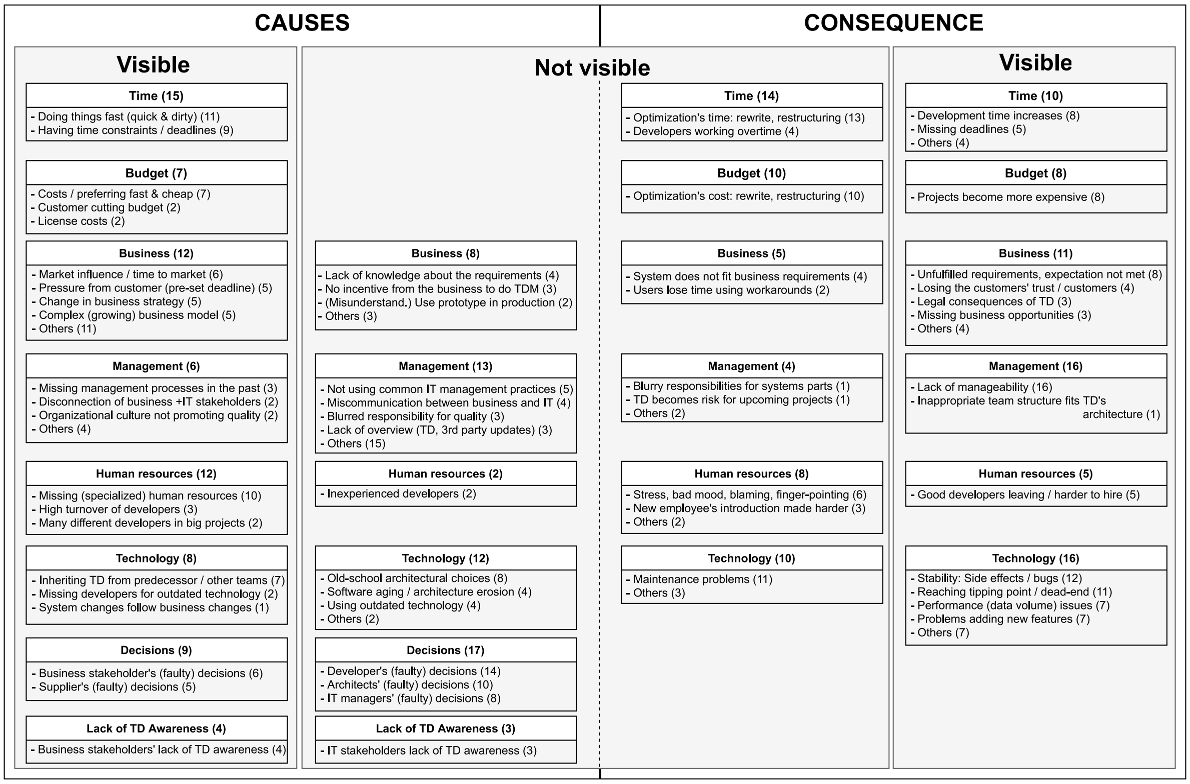
CAUSES
CONSEQUENCE
Visible
Not visible
Visible
Time (15)
- Doing things fast (quick & dirty) (11)
- Having time constraints / deadlines (9)
Budget (7)
- Costs / preferring fast & cheap (7)
- Customer cutting budget (2)
- License costs (2)
Business (12)
- Market influence / time to market (6)
- Pressure from customer (pre-set deadline) (5)
- Change in business strategy (5)
- Complex (growing) business model (5)
- Others (11)
Management (6)
- Missing management processes in the past (3)
- Disconnection of business +IT stakeholders (2)
- Organizational culture not promoting quality (2)
- Others (4)
Human resources (12)
- Missing (specialized) human resources (10)
- High turnover of developers (3)
- Many different developers in big projects (2)
Technology (8)
- Inheriting TD from predecessor / other teams (7)
- Missing developers for outdated technology (2)
- System changes follow business changes (1)
Decisions (9)
- Business stakeholder's (faulty) decisions (6)
- Supplier's (faulty) decisions (5)
Lack of TD Awareness (4)
- Business stakeholders' lack of TD awareness (4)
Business (8)
- Lack of knowledge about the requirements (4)
- No incentive from the business to do TDM (3)
- (Misunderstand.) Use prototype in production (2)
- Others (3)
Management (13)
- Not using common IT management practices (5)
- Miscommunication between business and IT (4)
- Blurred responsibility for quality (3)
- Lack of overview (TD, 3rd party updates) (3)
- Others (15)
Human resources (2)
- Inexperienced developers (2)
Technology (12)
- Old-school architectural choices (8)
- Software aging / architecture erosion (4)
- Using outdated technology (4)
- Others (2)
Decisions (17)
- Developer's (faulty) decisions (14)
- Architects' (faulty) decisions (10)
- IT managers' (faulty) decisions (8)
Lack of TD Awareness (3)
- IT stakeholders lack of TD awareness (3)
Time (14)
- Optimization's time: rewrite, restructuring (13)
- Developers working overtime (4)
Budget (10)
- Optimization's cost: rewrite, restructuring (10)
Business (5)
- System does not fit business requirements (4)
- Users lose time using workarounds (2)
Management (4)
- Blurry responsibilities for systems parts (1)
- TD becomes risk for upcoming projects (1)
- Others (2)
Human resources (8)
- Stress, bad mood, blaming, finger-pointing (6)
- New employee's introduction made harder (3)
- Others (2)
Technology (10)
- Maintenance problems (11)
- Others (3)
Time (10)
- Development time increases (8)
- Missing deadlines (5)
- Others (4)
Budget (8)
- Projects become more expensive (8)
Business (11)
- Unfulfilled requirements, expectation not met (8)
- Losing the customers' trust / customers (4)
- Legal consequences of TD (3)
- Missing business opportunities (3)
- Others (4)
Management (16)
- Lack of manageability (16)
- Inappropriate team structure fits TD's architecture (1)
Human resources (5)
- Good developers leaving / harder to hire (5)
Technology (16)
- Stability: Side effects / bugs (12)
- Reaching tipping point / dead-end (11)
- Performance (data volume) issues (7)
- Problems adding new features (7)
- Others (7)

Fig. 4. TD causes & consequences visible and not visible to business stakeholders (Number of managers mentioning this topic in brackets). For details on "Others" see additional materials.

Technological causes. This category comprises causes related to the evolution of various technologies and to growing knowledge, e.g., regarding best practices and solution architectures. (**Visible:**) Many IT managers report inheriting TD from previous projects, e.g., *"the problem is that these systems in our company, they live a long time and the first versions were created when my person was crawling [as a baby] there and I did not have too much influence on what was happening here"*. This is usually understood as a problem by the business stakeholders, e.g., if two companies merge, problems are expected. (**Invisible:**) This category also includes the reports of missing technology in earlier times: *"[I]f we now go back a bit to the 90s, then there was this whole issue of identity management. There were no real solutions, or only very, very expensive solutions"*.

Time-related causes. This is another category clearly influencing TD. While time pressure is often caused by business decisions, as explained earlier, this category additionally comprises other time-related causes. IT stakeholders perceive overwhelming time pressure in the IT industry, which could lead to always choosing the fastest solutions even when there is no direct time pressure. (**Visible:**) Various examples of this overwhelming time pressure in the IT industry were given in statements like *"there just wasn't time to do anything else"*, or *"time is much more important"*, (**Invisible:**) We did not find invisible causes in this category as time constraints can usually easily be explained to business stakeholders.

Human resource causes. This category encompasses an overall lack of IT specialists, the absence of specialized or knowledgeable developers, and the high turn-over rate many companies suffer from. (**Visible:**) An example of specialists leaving the company was: *"We can also say that it [a TD item] is maybe due to a shortage of people in the DevOps department. Because we used to have two guys on the team that had different responsibilities. Now we are left with only one"*. (**Invisible:**) A lack of developers also leads to a situation where inexperienced developers implement substandard solutions. The following is explained by an IT manager who used to be a developer himself: *"[F]rom my own evolution, my own view of these things, I see that experience makes you appreciate certain aspects that in the short term are not important"*.

Budget-related causes. This category comprises funding–related causes like cutting a project's budget, license costs, or the general preference for cheap and fast solutions. (**Visible:**) One IT manager gives an example of a customer's statement: *"I simply can't afford it now to change everything from some Windows Forms application to some new web design"*. (**Invisible:**) We did not find invisible causes in this category as budget constraints are commonly discussed with business managers.

Lack of TD awareness. This category includes causes related to various decision-makers being unaware of TD or of the consequences a TD item might have. (**Visible:**) A focus group participant summarizes that the lack of awareness may be a problem: *"I think what is really important here is to create awareness on a management level"*. (**Invisible:**) The lack of TD awareness on a developer's level may lead to a lack of TD identification, e.g.: *"[S]o the disadvantage may be the lack of detection of everything, the lack of awareness that something is technological debt, which in fact may have some negative consequences"*.

3.1.7. Technical debt consequences

We discovered a set of TD consequence categories (number of cases in brackets): Technological consequences (96), Loss of time (75), Management consequences (46), Business consequences (46), Additional cost/budget consequences (41), and Human resource consequences (26).

Similarly to the causes, we give one example of visible and invisible consequences for each category, respectively. More details are in the additional material (Wiese and Borowa, 2022).

***Technological consequences*.** These are the consequences usually easily noticeable by the developers, particularly, maintenance problems and architectural erosion. Furthermore, some technological consequences directly impact the business and can be perceived by business stakeholders, too, e.g., unstable systems, development reaching a dead-end, problems adding new features, or performance issues. (**Visible:**) In some extreme cases, system parts had to be turned off completely, as one participant told us: *"[T]hen we discovered that we have to extract it [TD] for a couple of days, weeks. And until then, we have to disable the feature because it simply doesn't work, and it slows down the entire system"*. (**Invisible:**) Issues related to working with outdated technology are another example of this category: *"So that means using outdated technology in some form that makes it difficult for us to keep systems up to date"*.

***Time-related consequences*.** The consequences of this category are commonly the time overhead needed to add new features or fix specific errors. (**Visible:**) One IT Manager explains the impact this time overhead may have on the implementation of new features *", where we sometimes really have to wait a year and a half until they [the changes] can finally be implemented"*. (**Invisible:**) A not well-researched issue resulting from the loss of time is that developers might have to work overtime: *"The expectation is that such changes can be done in a very short period of time. It turned out that it is not [possible]. Then, of course, additional stress and overtime work are standard"*.

***Management consequences*.** This category comprises the impact TD might have on the management of projects or project pipelines and the risk TD poses for IT management. (**Visible:**) Many IT managers described that TD *"hits you hard and surprisingly"*, e.g., *"if you haven't been aware of these things before, then at some point things hit you . . . to a great extent and usually at an inopportune time"*. (**Invisible:**) In one case, an IT manager described that the organizational structure of his company developed analogous to the debt-ridden architectural structure. This led to a situation where *" we had to rethink the underlying architecture in order to be able to say: Okay, certain teams are only responsible for certain areas"*.

***Business consequences*.** These consequences affect the business directly like expectations about features that were not met or losing the customers' trust. (**Visible:**) An example for a visible cause is a missed business opportunity: *"So every minute that I have to run after an undocumented feature that went wrong somewhere unexpectedly, that not only costs time, but it also prevents me from developing anything forward where I can somehow make more money with it"*. (**Invisible:**) Systems created but not used by the business due to a missing business fit might be invisible to the business stakeholders. *"Sometimes you have applications that are standing there, that hardly anyone uses anymore, but you still have to apply security patches, releases somehow. [They] at least inherently by being there, simply create certain risks"*.

***Budget-related consequences*.** The budget is influenced by TD when TD had to be repaid, e.g., by a rewrite. Furthermore, projects might get more expensive, e.g., if additional licenses are needed. (**Visible:**) A perceivable consequence is that *" the development costs had to be gradually increased higher and higher, and then the consequence was the cost of rewriting this system"*. (**Invisible:**) The simple need to rewrite a code fragment is an example of an invisible consequence: *"[TD is] a type of debt, as a result of which some additional costs are incurred . . .. [S]uch a debt has to be repaid, let's say in the form of rewriting a fragment of the code or, in the worst case, by rewriting the entire code"*.

***Human resource consequences*.** is the final category, including influences on the development team members and a company's staffing levels. (**Visible:**) A major problem related to human resources was that developers were leaving or harder to hire. *"Outdated technology doesn't attract developers either. That is, if I'm working with ancient technology now and trying to hire people"*.

(**Invisible:**) Furthermore, TD can lead to stress, bad developers' morale, and blaming each other. An IT manager complained about *"the process of blaming each other, both sides, or crying about who was at fault and why, how angry everyone was with each other"*.

3.2. RQ1.2: Current technical debt management

We explored the current TDM process according to IT managers along the TDM activities: TD Awareness, TD Incurrence & prevention, TD identification, TD monitoring, TD measurement, and TD prioritization & repayment.

Making conscious and rational decisions about incurring and repaying TD is a crucial part of TDM. Conscious decisions about TD incurrence may lead to the prevention of TD items. Conscious decisions about TD repayment are related to prioritizing TD repayments. Therefore, we address the decision-making processes of the IT managers in Section 3.2.2 and Section 3.2.6.

3.2.1. TD awareness levels

We identified the TDM knowledge and awareness of the IT managers' teams with regard to Kruchten et al.'s TD awareness levels (Kruchten et al., 2019). Most IT managers worked at companies on levels 2 and 3 of these TD awareness levels, i.e., the IT managers knew the concept, and knew about their own TD, but were missing methods to manage TD **(level 2)** and had problems communicating these TD items to business stakeholders **(level 3)**.

Only one manager mentioned that they had never heard of the term TD before **(level 1)**: *"What is technical debt? To be honest, I've never really thought about it . . .. I'm going to be a very poor respondent here, unfortunately, but I don't think [I know] anything"*. However, after explaining the definition to them, they replied: *"Of course, this is a familiar topic"*.

One top manager pointed out a TDM process integrated into Enterprise Architecture Management **(level 4)**. This manager explains: *"[W]e have a very good repository through our enterprise architecture management, where we have documented our IT landscape and from a wide variety of perspectives . . ."*. He further reports: *"There [are] . . . regular board meetings. We call it the IT Architecture Board. This is led by our enterprise architect, who then comes in with prepared decision documents. He coordinates or works on these with his respective architects in the various specialist domains . . .. And in the course of this, of course, this topic of technical debt is also addressed"*.

3.2.2. TD incurrence & prevention

Oftentimes TD was caused or increased by reckless or simply bad decisions (see Section 3.1.6. We also assume that mistakes are unconscious decisions to do something in an incorrect way. Decisions that seemed to be valid in the beginning might turn out to be lacking after a period of time. Finally, decisions made by uninvolved stakeholders, e.g., suppliers or government, can turn out to be bad for a specific system.

To analyze this decision-making process and the reasons for TD-related decisions, we examined the reasons why IT managers were willing to incur TD and what led them to refrain from incurring TD.

Why to incur TD?. The incurrence can happen unconsciously or consciously, which was mentioned nearly equally often by the IT managers.

Firstly, they reported situations where TD was incurred **consciously** after evaluating the consequences of TD versus the benefits, e.g., *"we discussed various options, and there was always a fast track. But maybe the solution is not very good, maintainable. . . . But then we said: "Yeah, but we want to deliver, but we will do the high-end solution later on"'*.

Secondly, another conscious incurrence is the need for a quick-fix of a bug, e.g., *" So let's say with the bug fix, for example, I have a problem that I need to solve. Then I might do a quick&dirty bug fix so that the problem is solved."*.

Thirdly, one participant of the focus group explained that they incur TD consciously but already plan the repayment. *"We [are] creating things very fast and [do] not talk [negotiate TD incurrence]. . . . But what we do then is to plan a road map in order to constantly create proper data models in the background . . . so that we can shift later on . . . but the end-user already has the value"*.

Fourthly, **unconscious** TD incurrence was mostly mentioned by the IT managers as caused by a lack of awareness of TD consequences, i.e., TD items are incurred *" not necessarily with adequate preparation or anticipation of the consequences that might come as a result. "*

Finally, TD was also incurred unconsciously, when there was a lack of better options in former times, e.g., *" if we now go back a bit to the 90s. . .. There were no real solutions, or only very, very expensive solutions . . . and in this respect, we started to build small solutions [ourselves]. "*.

Why not to incur TD?. Firstly, the awareness of TD consequences is the main reason not to incur TD, as stated by one IT manager: *"I mean, you can minimize the risk of [TD's] occurrence in the sense of promoting an appropriate technical culture"*

Secondly, a conscious TDM might lead to a raised awareness and change the mindset of stakeholders. This, in turn, might lead to the prevention of unconscious TD. One manager explains: *"It is necessary to have discipline in the sense that just when there is no factor that forces us to do something very quickly, we should think about how to do it well so that the debt is not incurred"*.

Thirdly, TD can be prevented if the debt-free option is also the better solution from the business perspective. This was mentioned by an IT manager talking about a company merger and the resulting IT merger: *"I can bring that together by – kind of – doing a layer on top that glues things together. Or I have different teams do the reporting and then the reporting has to somehow be layered on top of each other in Excel"*.

Fourthly, one manager mentioned that their company only prevents TD in central components they call "tools": *"Tools yes, because we work with tools every day and ninety people use this editor of ours. So tools yes, [but] everything else [is] no longer [prevented]"*.

Lastly, the focus group discussion revealed that to avoid TD, business stakeholders should not decide on TD incurrence. *"[U]sually at least a businessperson without IT background will have no idea [of TD]. And usually, . . . they don't care at all if there is technical debt or not. Because, usually, their mindset is not what will be in one year with this product but like: 'I need these numbers now – yesterday, and it's great that they're there, and I don't care [about next year]"*.

TD prevention strategies. Regarding TD prevention strategies, it is striking that 15 managers mentioned having implemented at least some TD prevention procedures, e.g., by making rational decisions regarding technologies, architecture strategies, or when choosing a supplier. An example of the latter is: *"we do not use anything open-source, and if we already use something open-source, we have the support from some external provider"*. Moreover, they enabled TD prevention mechanisms regarding code debt by establishing (good) coding practices, e.g., unit testing, code review processes, or *"measuring the quality of the code through linter, through static code analysis"*.

3.2.3. TD identification

TD identification by chance was the most frequently mentioned way of identifying TD, e.g., *"So, it's usually in the area in which they [the developers] have likely developed features because that's where they have a look. And then they figure out that something could be done a lot better"*.

While the IT managers agree that early identification is relevant for the manageability of TD, they generally did not have any identification strategies and conceded that they usually discover TD items "too late", e.g., *"That was then actually already too late. . .. These measures were so big that you can no longer just talk about refactoring, but it was really a change in architecture and that was a really big rebuild"*.

3.2.4. TD monitoring

Regarding TD monitoring, many of the IT managers mentioned that their teams create TD tickets for each TD item, but nearly no managers had an overview of these tickets, let alone the ability to control these. Moreover, they concede having problems with *"cluttered backlogs"* and *"rotting tickets"*. One IT manager describes: *"And then they [the tickets] usually disappeared in the backlog [–long pause–] until they were deleted sometime after two years"*.

For this reason, most of the IT managers agreed that someone has to keep track of the TD items on a regular basis to create a TD overview, e.g.: *"In fact, it is also the case that we regularly . . . carry out assessments after certain periods of time"*. They agreed that a responsible person must be tasked with keeping track and warning about the potential incurrence of TD in decision-making processes, e.g.: *"[W]e would have to have someone who basically keeps track of these things and says: Watch out, we're in danger of disaster here"*. Some mention that they expect the team's architect to be this person, e.g., for the "level 4" manager, this *"is led by our enterprise architect . . ."*. While the participants all agreed on this, most of them did not have established a process or responsible person for keeping track.

3.2.5. TD measurement

IT managers conceded to having problems calculating precise values describing aspects of TD, e.g.: *"[T]here are still these options related to estimating the cost of this debt, interest, this type of thing, but it's probably not [easy] like that. No one will be able to estimate it"*.

Still, the estimation of the repayment's business value is relevant for prioritizing TD items: *"So, you know, there is always something not correct right now. And in the end, it's also about evaluating what is the relevance of that right now. And when do we need to take care of it?"*

In this regard, some managers mentioned that the accuracy of the estimation values is secondary to the time needed to carry out these estimations. One manager suggested using *"a risk factor that makes it clear to me how important that is. And just such a, let's say, complexity assessment in story points. Preferably in story points, because then I can compare it with the stories themselves"*. An idea that emerged from one interview was to use architects to get a valuable estimation: *"Surely, there should be people in the organization who are able to quantify the impact in some way, but this, in turn, is quite difficult. I think that people who have this kind of thinking are at the level of architects"*.

3.2.6. TD prioritization & repayment

Analogous to the TD incurrence and prevention, we identified reasons that lead IT managers to repay TD items and that lead them to refrain from repaying these.

Why to repay TD?. Firstly, regarding the repayment of TD, most IT managers described that repayment was possible after some kind of tipping point was reached. These tipping points can be vague, e.g., *"at the moment when such a debt begins to be burdensome, you have to get rid of it, rewrite, for example, such fragments"*. Other tipping points are more distinct, e.g., *"because the report generator doesn't exist anymore, because it's not maintained anymore. Something like that. So where basically a technological change forces a redesign on us"*.

Secondly, sometimes the evaluation of benefits and drawbacks led to some repayment activity. For example, one IT manager explained how the old architecture with high amount of business logic on the database level led to performance issues, and *"to work around the problem, additional resources were added to this database, . . . but this resulted in costs because you had to buy additional licenses . . .. So there was, you could say, interest on this debt . . .. At one point, it was decided that it had to be optimized and partially rewritten to the application layer, and that's how it [the TD] was paid off"*.

Thirdly, some TD was repaid as a result of a single person's initiative, i.e., *"there were people who were just implementing features and they saw that there was code around that needed refactoring, so they refactored it and reduced the technical debt"*

Finally, the planned repayment after conscious incurrence was mentioned, e.g.:"[I]f we introduce something that's just known to be bad but easy to do and that's the only reason we're doing it, we immediately plan a quick fix for the next sprint or in the near future . . .".

Why not to repay TD?. Firstly, by far, the primary reasons not to repay TD were other priorities, including time and budget, business value, or new features. One IT manager stated his experience: *"I have faced throughout my previous experience working in different departments that some systems are being evaluated like, 'yeah, it looks really crappy from a technical perspective, but money-wise, we are kind of fine and we do not want to invest in renewing this stuff.' "*

Secondly, reasons that led to a lack of TD repayment are management mistakes, e.g., one IT manager conceded: *"What – I think – I totally underestimated from a management perspective, actually, how stubbornly certain technical debts carry into the future"*..

Thirdly, problems distributing developers to TD topics became a problem, i.e., *"the technical debt is much deeper and a lot harder to handle because some refactoring things are so, so complicated or so expensive that it's hard to find volunteers that would gladly do this"*.

Finally, there are organizational challenges, which might prevent TD repayment. One IT manager talked about TD, which had to be repaid across teams, and postponed the repayment for a long time. *"And then it was passed on and ultimately decided on by all the teams and also designed: How do we approach this now and solve it holistically and not only extinguish fires within our area"*.

TD repayment strategies. Regarding TD repayment strategies, the IT managers mentioned various ideas for repayment strategies, e.g., regularly allocated time slots or repayment projects. An example of a TD repayment project is the following: *"They are planning now a tech agenda, and they also create projects which are dealing with working on technical debts like this [project] we are now doing"*. However, the implementation of these ideas was rarely successful in the long term, culminating in the statement: *"Someone eventually gets annoyed and starts doing it"*.

This leads to the tendency that "radical measures", e.g., starting from scratch or abandoning a project completely, were the most frequently mentioned repayment strategies. For example: *"[Then] a company-wide decision has been made that we are doing a re-engineering. Basically, not even a re-engineering, but we did it all over again"*.

Iterative repayment was only mentioned a few times and mostly in terms of migrating to a new system iteratively, e.g., *"but there's a plan which has been implemented for two years now, only migrating in stages"*.

3.3. RQ2: The IT managers' main concerns regarding technical debt

As a result of clustering the themes that evolved from coding, we identified three major clusters, i.e., concerns of the IT managers, which are described in detail in the sections below. These concerns are communication about TD (Section 3.3.1), the establishment of a TDM process (Section 3.3.2), and dealing with what we call vintage systems (Section 3.3.3). The results of this clustering were presented to the focus group to discuss and accept or reject the hypotheses. The insights of this discussion are added to the respective subsection.

3.3.1. Communicating about technical debt

IT managers are in the position of a mediator between IT and business stakeholders. Thus, their communication issues are twofold and include communication with business stakeholders and developers.

Communicating with business stakeholders. IT managers often seem to lack the business stakeholders' trust and understanding regarding TDM. However, they can only manage TD if the TDM process is accepted and financed by the business stakeholders. Many IT managers mention intense negotiations with business stakeholders about the business value of TD repayment, e.g.: *"So I had to convince the [business] managers in the organization to take a serious approach to solve these problems, not to act on the basis of just patching things up"*.

However, they are aware of the difficulties developers face in providing such specific information about a business value. One IT manager saw it as their duty to mediate in this case: *"And everything is measured, so to speak, by business value and less by technical value and that is then in turn for my area a challenge to argue why we have to do certain things"*.

When trying to explain the TD concept to their business stakeholders, IT managers were often missing appropriate methods, e.g.: *"Most of the customers we have are IT illiterate. No IT department of their own. We actually work with amateurs"*.

The focus group agreed with this concern and reported similar problems, even before this specific concern was presented to them as a point for discussion by the researchers.

Communicating with development teams. The IT managers described two potentially problematic kinds of developers: quality-oriented developers with a great interest in high software quality, and speed-oriented developers with less focus on quality.

The concern that arises with quality-oriented developers is what we would like to call **"developers crying wolf"** in reference to the fable of "The boy who cried wolf" (Aesop, 2002). These developers tend to complain about TD items regardless of their impact on the business, i.e., the effectiveness of the TD item (Schmid, 2013). For example, one manager lamented: *"[E]veryone who gets someone else's software says: it has to be written from scratch"*.

As a result, IT managers may become desensitized to the term TD, and developers may not be heard when a real crisis occurs. *"The developers also say: So how many times do I have to tell you* *that this is a big problem? Does the shit really have to hit the fan?* " This makes it hard to identify and prioritize relevant TD items.

The second type of developer abides by **"it works"** and **"just do it"** principles. They have no habit of discussing solution options, prefer to work alone, do not question requirements, and refrain from voicing their concerns. Many managers mentioned this as a problem, e.g.: *"It didn't matter [to the developer] that it was all duct tape there, but it was working".*

Many IT managers in our study mentioned that regarding TDM, particularly TD prevention, they see it as their task to change these developers' mindsets. One IT manager explained: *"My idea was also to create people [nurture and educate employees] who are able to take care of these things, to educate developers so that they also care".* Yet, the IT managers are missing processes on how to change these mindsets successfully, as stated by another IT manager: *"And on this mindset – I only have a certain degree of influence on it. . .. I can't stand next to it all the time, micro-manage, and have everything presented to me".*

The focus group could relate to both developer types and the related concerns. Regarding the "developers crying wolf", one participant mentioned that particularly experienced developers are *"creating like a tunnel vision, thinking mostly about the technical stuff".* Regarding the "it works" type of developer, another focus group participant argued: *"But we, as technical experts need to, like, really suggest the solution and not just be executors of demands".*

3.3.2. Establishing a technical debt management process

As explained in Section 3.1, most IT managers knew the concept of TD, and knew about their own TD, i.e., they had at least some level of TD awareness. However, most IT managers had not established methods to manage TD properly. They had already implemented some measures for TDM, e.g., filing TD tickets or having a percentage of time for TD repayment. Nevertheless, they failed in their attempts to manage TD sustainably for various reasons.

In Section 3.2, we identified the IT manager's current knowledge and their implementation of TD activities. In the following subsections, we identify three (sub-)concerns related to the TD activities that contribute to the main concern of establishing a TDM process.

TD awareness, incurrence, and prevention. Most IT managers identified that a lack of awareness of a TD item's consequences leads to unconscious TD incurrence, i.e., TD items are incurred *" not necessarily with adequate preparation or anticipation of the consequences that might come as a result".* Furthermore, the awareness of TD consequences was the main reason to avoid TD, as stated by one IT manager: *"I mean, you can minimize the risk of [TD's] incurrence in the sense of promoting an appropriate technical culture"* Therefore, the awareness of TD is a major concern that needs to be addressed for a successful establishment of a TDM process.

TD identification and monitoring. Almost no IT manager had an overview of their TD items, although they admitted that TD monitoring is essential, e.g., *"There is this saying: 'If you can't measure it, you can't manage it.' So if I don't have an overview of what problems I actually have, then I can't tackle them.'* Their backlogs were cluttered with TD items but no structure was set into place to gain an overview of them. Furthermore, while most of the IT managers agreed that someone has to keep track of TD on a regular basis to create a TD overview, they did not assign a specific person to this task. *"[W]e didn't have a formalized method, we generally had a message to have 20% [of our time for TD repayment], but we didn't have control over it".* The derived concern of the IT managers is to structure their backlog, create an overview, and keep it up-to-date.

TD measurement, prioritization and repayment. As shown in Section 3.2, the estimation of the repayment's business value is relevant for prioritizing TD repayments and for communication purposes, i.e., counteracting the "crying wolf" problem, and negotiating with business stakeholders. However, all IT managers lacked information about TD measures to decide on the most urgent TD items before they became too big to handle properly. *"At most, you can somehow indirectly assess that this technical debt exists and in what dimension, but I think it's largely indirect . . .. [T]there are no clear indicators beyond the already glaring situations that help quantify this debt".* Regarding a business value, some managers mentioned that high accuracy of the estimated values was not crucial and that easy-to-use methods that would fit into their existing set of tools and methods were more essential. The struggle with a decision on TD repayment was explained by an IT manager: *"Sometimes these are very small things that originally cost or would have cost 20 days of developer effort. And years later, I'm standing there like this: Oh yes, to rebuild that now, but that costs 200 days, so ten times as much".* One manager described this decision-making process as a *"kind of art of making a choice whether it's worth devoting time to it and whether it will pay off in some sense".* In this case, the main concern of IT managers is to estimate as effortlessly as possible the negative impact of a TD item on a system and the positive contribution of its repayment to a business value.

The focus group agreed with the observations of all three clusters and suggested that TDM should be perceived as a part of risk management, in order to be better aware of the systems' TD. Moreover, to avoid TD, the focus group advised that business stakeholders should not decide on TD incurrence, *"because, usually, their mindset is not what will be in one year with this product but like: 'I need these numbers now – yesterday, and it's great that they're there and I don't care [about next year]".*

3.3.3. Dealing with vintage systems

Many IT managers noticed TD because of problems with outdated and very old systems with deep-seated ATD, which they refer to as systems or decisions *"of the 90s".* Particularly, out of this study's six top managers, four mentioned these 90s systems and software aging as being the main source of TD. While these 90s systems have manifold problems, they have made positive contributions in their time. One top manager summarized: *"I have to say that sometimes I talk as if what we have as software is somehow a big disaster. . . . On the other hand, you have to look at it: The software has been running for 25 years and that is a proud age. . . . The people who developed it must have done a lot of things right, otherwise, it would have gone down the drain a long time ago".*

While the term **"legacy system"** was originally used for old and outdated systems and technologies (Bennett, 1995), nowadays, the term is often used differently in practice. Any system that can no longer be supported and updated may be called "legacy" regardless of age (Khadka et al., 2014).

Therefore, we prefer to call old, valuable systems **"vintage systems"** in an analogy to vintage cars. These old systems, just like vintage cars, have survived more than 20 years, which is a great engineering accomplishment. Thus, vintage systems are a subcategory of legacy systems. They are a source of great business value and a company may even decide to keep using and improving them. Fitting the analogy, there are communities of lovers of these old systems, just as with cars, e.g., "Legacy Code Rocks" (Anon, 2022c). Engineers skilled in working with these vintage systems and cars are both hard to find and expensive.

Despite their value, the IT community and businesses may have to accept that there is a maximum age for which systems, as well as cars, can operate efficiently. The older the system or car gets, the more maintenance is necessary and it is usually not efficient to keep such a car or system. This can sometimes cause old-established companies to fall behind younger companies, since there is so much TD in their systems, that it cannot be repaid within a reasonable period of time. One top manager mentioned in relation to a vintage system: *"[S]omeone was then assigned to investigate that and he then comes back with well, software license costs, I say, in the seven-figure range. And by the way, it takes three years. So you sit there as the responsible IT manager and then you think: I can't imagine that right now"*.

One of the IT Managers' concerns seems to be a lack of **repayment strategies** for hard-to-tackle ATD typically present in vintage systems. They miss feasible architecture refactorings in small increments. Their usual way of dealing with this kind of TD is one of the following: (1) rewrite the whole system (*"Changing anything was very difficult, and generally we had to . . . write everything from scratch"*.), (2) switch it for a third-party system or (3) get rid of the old system completely (*"So you could say we removed the debt by removing the tool"*.).

Another critical concern with vintage systems is **knowledge vaporization**. The valuable knowledge that older developers have about vintage systems makes these developers irreplaceable. If they leave, retire, or die, their knowledge might be lost. Moreover, these developers might have the "vintage mindset" to do what they are told, not to question authority, and not to share their knowledge freely; as one manager put it: *"People have only a limited interest in making a know-how exchange on a level, where they then make each other more clever"*.

This concern was one that the focus group participants could not relate to. This may be due to their company's relatively young age. They suggested to separate the TD research on younger companies, e.g., start up's, from old-established ones.

3.4. RQ3: Improvement measures for technical debt management

Some of our questions aimed to identify ideas from IT managers on how the situation regarding TD could be improved. Thus, we asked questions to identify the potential for improvement in the TDM process (Section 3.4.1), and the communication related to TDM (Section 3.4.2). We did not ask for potential improvements related to vintage systems, since this concern emerged later during the analysis.

3.4.1. Technical debt management

In Section 3.2, we present the IT managers' knowledge about TDM and, in Section 3.3.2, we show their concerns regarding the establishment of a TDM process. In this section, we present the TDM process' potential improvements from the IT managers' perspective. For this purpose, we asked IT managers about their role in the TDM process, including the actions they can take to influence and improve the process, and the benefits and drawbacks they expect from implementing a TDM process.

IT manager's role in the TDM process. Firstly, all IT managers saw it as their role to manage TD. One IT manager summarized: *"Basically, I'm responsible for the technical state of the platform, so I'm also responsible for the technical debt"*. Secondly, they state to have the authority to manage TD, e.g., *"[I] did have the influence to put these corrective actions in place"*. Finally, they conceded that they should have an overview of TD, e.g., *"I suppose also my role would be to have an overview of things. That cannot be too specific, but then I should encourage people that are in the team to have an eye out for everything"*.

TDM process benefits. Most IT managers saw the benefits of a TDM process primarily in terms of manageability of TD and their systems: *"[T]o me it just brings . . . a manageability to the whole thing . . .. [I]f you don't take care of it and you don't have an overview of what the problems are, you can't reduce them"*. A manager pointed out: *"It's important for me as a manager to have an overview in order to be able to set priorities"*. Another manager focused on predictability: *" If we only discover the technical debt when we approach something and realize now we don't need one hour to do something, but 20 h, because we've reached a dead end. Then, of course, we always have a resource problem"*. Other IT managers have not even considered the benefits of a TDM process before: *" When the system runs well and the team is happy, then pretty much my job is well done. I honestly never, never thought of this question. "*

TDM process drawbacks. While the benefits were self-evident, all managers had problems giving examples of any TDM drawbacks: *"[You] definitely have to do it [TDM]. I don't know any disadvantages. I just know – Frankly, I don't know – So what should be a disadvantage? That you put time into it?"*

The most frequently mentioned drawback was the effort, i.e., time and budget that needs to be put into such a process: *"[I]t would take people who could do something else at that time, ... That's probably the only thing that comes to mind"*. However, most IT managers also mentioned immediately that this effort would be justified by the benefits they could gain from such a process, e.g.: *"Well, it costs: You have to sit down, of course, and it just takes a bit of time. But I would say that is negligible"*.

After some thought, some IT managers pointed out that it is difficult to obtain the budget for TDM from their customers, e.g., *"The only exhausting thing, I would say, is that at the end of the day - especially with our customer clientele - it's hard to get money for it"*.

Other topics mentioned as potential drawbacks by a few IT managers were the risk of over-engineering and micro-managing. Regarding over-engineering, one IT manager explained: *"[D]evelopers are also interested in 100% solutions. I can understand that they want it to be perfect, but then you have to manage away this over-engineering character a bit.*" Regarding the danger of micro-management, another IT manager stated: *"It just gets a little bit of bureaucracy out of it . . .. [P]ulling away from the current thing that needs to be patched up immediately"*.

The particularly interesting topic of TD becoming overwhelming and leading to a loss of motivation was mentioned by two IT managers: *"Another challenge may be, . . . that when someone sees how much [TD] there is to do, they lose motivation to do anything at all"*.

Three other disadvantaged were mentioned once each: (1) TDM might not be worth doing in small and short projects, (2) TD items are cluttering the backlog, and (3) TDM creates communications problems. The last topic was explained as follows: *"[TDM] puts additional demands on the development team and also on the project sponsor to understand the situation and if you try to do it in a situation where there's a lack of competence and a lack of understanding, it generates additional problems"*.

3.4.2. Communication

In Section 3.3.1, we showed the concerns IT managers have regarding communication with business stakeholders and developers. Below, we present the role IT managers believe in having regarding this communication topic. Furthermore, to identify potential improvements, we asked the IT managers what they need from the developers and business stakeholders, respectively, to improve TDM.

IT managers. IT managers interpreted their own role as being the mediator between developers and business stakeholders. Their role is to *"kind of try to negotiate between your tech team and business people, . . . because one side wants quick and fast and the other side wants the proper way"*. Some even mentioned the need to influence the company's culture regarding TD: *"Why does a company actually tolerate such technical debt? You often have corporate cultures that - especially in classical cultures - are not used to questioning things, questioning decisions, questioning orders, questioning solution proposals"*.

In their communication effort with the **business stakeholders**, some IT managers see it as their role to ascertain the business fit of the solution. *"One thing that is very important to me is that we do a solid project preparation. So where we do exactly these things like: goal, scope. What is in-scope? What is out-of-scope? "*

Moreover, they believe that it is their role to educate the business stakeholders on the topic of TD. One IT manager sees it as their task *" to create an understanding for this [TD], to get such discussions going with the management as a whole, i.e., to explain this to the management"*. Explaining the issue of TD is necessary to negotiate with business stakeholders to obtain resources for TD repayment, which is another major role of IT managers regarding TDM, i.e., *"granting the freedom also to reduce these technical debts is then up to the management, i.e., to make the time available.'*

In their communication with the **developers**, IT managers perceive their role as mentoring, teaching, and incentivizing a TDM process, e.g.: *"The other thing that I might do is definitely an element of education . . . so that they [the developers] are more aware of this technical debt"*. Another way IT managers try to create this TD awareness is to *"start to manage that debt in general, which is to be aware of those gaps or things, that we need to do somewhere, that we've put off doing in the past"*.

Furthermore, they see it as their role to create an environment based on trust: *"[T]hey [developers] have to be sure enough of their place in the project or such, to feel supported, to bring such things out, to say them somewhere"*. In turn, IT managers have to trust the developers and their decisions: *"If you have . . . the confidence in the development that they are not twiddling their thumbs for weeks, but that something is really being changed, . . . then you also know how important it is to manage these things [TD]"*.

Developers. Firstly, IT managers wished that their developers would communicate better and earlier about issues that occurred. They wanted the developers to proactively voice their concerns and question decisions. They want them to *"come forward and not only focus on the small thing that they are working on without considering how it interacts left and right with something else"*. One IT manager emphasizes the need to escalate such issues: *"I have to train my employees . . . to recognize something like that and to escalate it in time"*.

Secondly, IT managers want the developers to take individual action when needed, i.e., do small refactorings regularly. One IT manager wanted the developers to consider: *"Okay, if I get past such and such corners that are difficult, why don't I clean them up right away?"* They further concede that this *"would mean, for example, from a project management point of view, plan a little time for refactoring, but above all this awareness on the part of the developers"*.

Lastly, an IT manager mentioned that they need the developers to provide the information necessary to manage TD. They mentioned costs, impact, and risk estimations, but also that the shortage of qualified developers can become a bottleneck: *"[M]aybe it's not bad to know – sometimes you have such expertise within the team – who of the developers, for example, can even tackle this"*.

Business stakeholder. Firstly, most IT managers wished for better communication of business decisions. They wished for the business stakeholders to be actively involved in projects, because then *" he can understand the problems and then he can make those decisions, whether we leave it or take the time to change it"*. IT managers want to be informed and understand the rationale behind business decisions, i.e., they need *"communication about changes before they happen. Not that I open the editor [with the requirement], and everything is different than it was yesterday"*. According to our participants, business stakeholders should provide *"a clearer roadmap of where we want to go, so that we can think forward at least a couple of months"*.

Secondly, one issue was the business stakeholders' trust in the IT managers. *"So I need trust from my manager. Trust that what we're tackling is important. That it's important that we now reduce this technical debt that we've decided to do [repay]. I need the confidence"*. Another manager shared a similar observation: *"I'm responsible for the implementation, and then let me be responsible"*.

Thirdly, IT managers want the business stakeholders to recognize the importance of TD and to better understand its consequences, e.g.: *"Well, and he [a business manager] also has just to recognize that [TDM] as one of the important factors of building a business. . . In particular, the lack of [TD] management will have a negative impact"*. The focus group discusses a situation where warnings from the development team regarding the data quality of a report were simply ignored. They warned the business stakeholders *" Hey, there is . . . this quick and dirty solution. Please don't use it, if you want to have reliable information"*.. But they observed that the business *"people don't care, because as long as they have something, they are fine with it"*.

Finally, IT managers needed proper conditions, particularly time, to enable TDM without recurring negotiations for each TD item. This time aspect is twofold. For one thing, this means time for the developers to repay TD, *"because I think that every employee, if he is told that he has, for example, 15% of the time, 10% of the time a day or a month to do some verification and so on, to complete such information, and to propose software updates, then he will be happy"*. Additionally, IT managers need time to manage TD properly, e.g., *"[W]ell, these are things again, like this technical debt, when I know that some things would be nice to do, maybe to implement [a TDM process], but I don't have time to do it"*.

4. Discussion

In this section, we discuss the results of Section 3 with regards to the IT managers' main concerns as identified in Section 3.3. These concerns are communication about TD, establishing a TDM process, and vintage systems. Regarding the communication about TD, we focus on communication with business stakeholders, while communication with developers is discussed as part of the "establishing a TDM process" concern.

4.1. Business communication

Both the misunderstanding of the TD metaphor and communicating on different visibility levels are possible sources of communication problems.

4.1.1. Metaphor

TD is a frequently, yet colloquially and ambiguously, used term in practice. However, the idea behind the metaphor, e.g., principal and interest, is often hidden from practitioners. They usually do not consider the financial metrics of the metaphor, like interest and principal. This indicates that the metaphor does not help with business communication, for which it was originally created

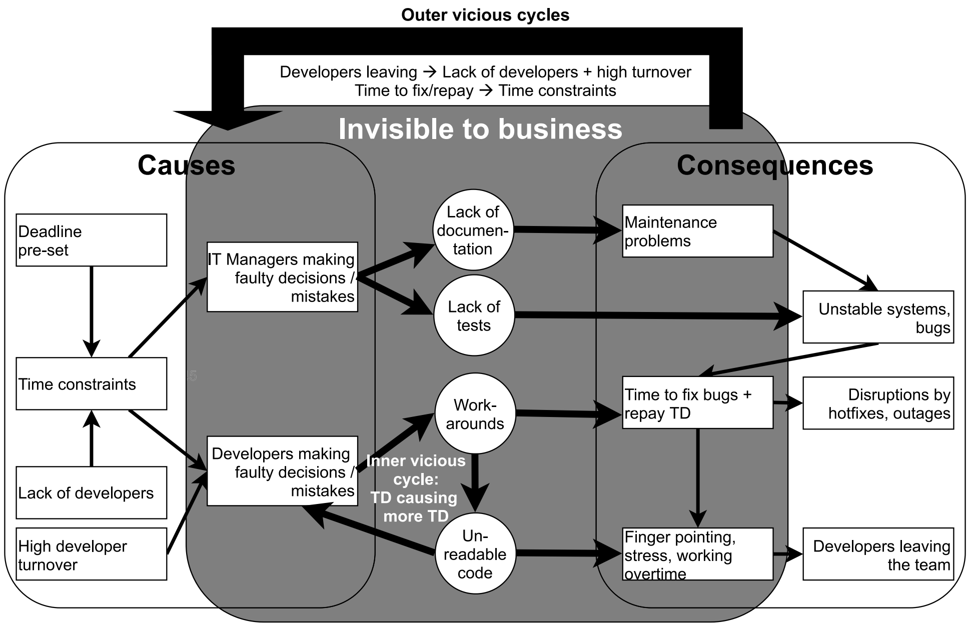
Outer vicious cycles
Developers leaving → Lack of developers + high turnover
Time to fix/repay → Time constraints
Invisible to business
Causes
Consequences
Deadline pre-set
Time constraints
Lack of developers
High developer turnover
IT Managers making faulty decisions / mistakes
Developers making faulty decisions / mistakes
Lack of documentation
Lack of tests
Work-arounds
Un-readable code
Inner vicious cycle: TD causing more TD
Maintenance problems
Unstable systems, bugs
Time to fix bugs + repay TD
Disruptions by hotfixes, outages
Finger pointing, stress, working overtime
Developers leaving the team

Fig. 5. Example usage of a 3-minute slide. (small arrows = chains of causes/consequences; big arrows = cause–effect relation).

for Cunningham (1992). Furthermore, IT managers invented their own metaphors and used them in business communication.

One IT manager suggested that a metaphor should be domain-specific to be understandable to business stakeholders. In retrospect, this was the case for Cunningham's TD metaphor that was developed while working in the financial sector.

Other IT managers suggested a kind of metaphor list or collection, from which IT stakeholders could choose an adequate metaphor. A starting list for such collection could be extracted from our results (see Section 3.1.2). An expanded list could be collected, e.g., via social media, though the usefulness of such a list would still need to be proven.

4.1.2. V4CTD model of visibility, cycles & chains of causes & consequences of TD

Through our V4CTD model's identification of invisible causes and consequences, we uncovered potential reasons for communication problems with business stakeholders. IT stakeholders often communicate using causes and consequences that business stakeholders do not understand. Sometimes developers argue by explaining the TD item itself, which might be invisible even to their IT managers. Yet, it is important to help business stakeholders understand the underlying dependencies of their own actions and the TD symptoms they may observe.

Firstly, it would be helpful if IT managers particularly asked their development team about these benefits and gave them guidance on how to prepare the information about these benefits. A top manager stated his mistake in underestimating the impact of TD, in particular, TD growth: *"That is the mistake. We should have worked out the benefits more precisely and said: 'What do we gain from all this?'"*

Secondly, our V4CTD model can help with the explanation of TD and argumentation for TD repayment. This kind of tool was even explicitly requested by one participant: *"How do I argue correctly? What does not serve as argumentation? A one-hour argumentation does not help. A manager usually has no more than a three minutes attention span".*

The V4CTD model can be used, to create a **"three-minute slide"** for an "elevator pitch": First, IT managers, with the help of their developers, may use the overview of in-/visible causes and consequences (Fig. 4) to look for their own chains and cycles of causes and consequences. Second, they can transfer these chains to the V4CTD model from Fig. 2. Finally, they can use this one-slide overview to explain and discuss TD with their business stakeholders.

In our example in Fig. 5, time pressure, and a lack of developers lead to unstable systems, business disruption, and developers leaving the team. Workarounds leading to unreadable code create an inner vicious cycle. Furthermore, unreadable code may lead to more developers' decisions to create workarounds. The lack of developers and developers leaving the team creates an outer vicious cycle.

4.2. Establishing a technical debt management process

Comparing the results of RQ 3 (Section 3.4.1) with the issues regarding TDM presented for RQ 1 and RQ 2 (Sections 3.2 and 3.3.2), we noticed a contradiction between their actual actions and their belief that they are responsible for TDM. On the one hand, most IT managers stated that TDM is an important topic. None of them saw sufficient disadvantages in TDM to outweigh its benefits (Section 3.4.1). On the other hand, only one IT manager's team used any kind of basic process to manage TD. The others did not have an overview of their TD, making TD unmanageable (Section 3.3.2). This may be an effect connected to the **status quo bias**, i.e., a cognitive bias that presents itself as an irrational preference for maintaining the current state of things (Kahneman et al., 1991). The status quo bias may make individuals more likely to choose the default option of not changing anything, particularly when a high number of alternatives is possible , which is the case when deciding on actions related to TDM.

Overall, IT managers are a crucial part of TDM. Establishing a TDM process and communication with business stakeholders is almost impossible without their support. Though all of the IT managers had some degree of control over time or budget, most of them did not connect their control of these two crucial factors with TDM.

To address the status quo bias, we propose researching how a TDM process might be successfully established rather than focusing solely on the details of the TDM process' structure and activities. To provide a starting point for this research, we identified three sub-concerns in Section 3.3.2, which also represents a sub-structure of closely related TD activities: (1) TD Awareness, Incurrence, and Prevention, (2) TD Identification and Monitoring, (3) TD Measurement, Prioritization and Repayment.

The **first sub-concern** is raising TD awareness to avoid the incurrence of unnecessary TD. Addressing this might be as simple as establishing a basic TDM process as a starting point, i.e. to overcome the status quo bias and "just get started". Some IT managers even decided to start a TDM process as a result of their participation in our study. An established TDM process might also help with the communication issues with developers (Section 3.3.1) as indicated by some IT managers (Section 3.4.2). Moreover, a structured and traceable process can give **"it works" developers** a framework for their work and help them understand and pursue quality objectives. The improvement measures that IT managers wish from developers (Section 3.4.2) may also be addressed through a TDM process as it could enable structured communication about TD. Finally, establishing a structured process might lead to a raised TD awareness not only for the developers but for the whole company, including business stakeholders. The (initial) effort needed for establishing such a TDM process is a drawback mentioned in Section 3.4.1. It is, therefore, essential to focus on introducing a TDM process in a manner requiring minimal effort.

Regarding the **second sub-concern** of creating a TD overview and keeping track of TD, the IT managers usually suggested the use of simple strategies (Section 3.2). Most recommended the use of already established ticket tools, e.g., Anon (2022d), Anon (2022e), or a wiki page to create a TD overview. One manager suggested that TD identification may take place during retrospectives. Other IT managers suggested including TDM in the Scrum process and using story points for TD measurements. Finally, an idea mentioned by two IT managers was the integration of TDM into risk management.

The **third sub-concern**, i.e., measuring TD to be able to prioritize TD items and to justify the repayment of TD, is the most difficult to solve. The idea of using financial metrics for TD, as the metaphor suggests, was not familiar to many IT managers (Section 4.1.1). However, most of them noted that a TD repayments' business value might help in negotiations with business managers (Section 3.3.2). A benefit IT managers have not considered is the positive influence an implemented prioritization mechanism might have on the issues with the **"developers crying wolf"**(Section 3.3.1). Analogous to the first and second sub-concerns, the IT managers mentioned that the measurements need to be easily calculated without putting additional (time) pressure on the developers. In return, they are prepared to accept the lower accuracy of these values.

Summarizing the sub-concerns, we presume that the TDM process and the identification of estimates must be part of an already established process in the company to enable a simple approach and to enforce the process in a sustainable way. Existing methods from academic research might be too disconnected from methods that are already used in practice, and further research might fill this gap.

4.3. Vintage systems

Vintage systems are a source of deep-seated TD, particularly ATD. When asked for the most impactful of their TD item examples, two managers decided not to choose the TD item that could simply be repaid by investing a sum of money. To them, this seemed to be an easy solution and, therefore, a less problematic TD item. Instead, they decided on a more complicated TD item that would not be easy to repay. *"One [TD item] is just an investment ruin, . . . you can solve that to some extent with money. [So let's focus on the other TD item.]"* This indicates that the worst cases of TD are those that cannot be repaid by a simple investment alone. These types of TD, which are usually part of intricate vintage systems, are extremely challenging to repay. IT managers responsible for vintage systems were often missing ideas or processes on how to deal with these problems. What is more, the great amount of accumulated TD might lead to the loss of motivation, as explained in Section 3.4.1.

We suggest that research should further focus on iterative approaches to migrate vintage systems without disturbing the business. Additionally, the research community could also explore methods that allow crucial parts of these vintage systems to live on and be useful for an even longer span of time. Finally, methods to sort through big amounts of TD might be helpful for IT managers when migrating vintage systems or, particularly, when avoiding a complete migration. There are similar methods for code debt in static analysis tools, e.g., Anon (2022f), but they are still missing on a general level.

However, companies with systems that are more than 20 years old need to be aware of the issues involved and implement investment plans to fund modernization projects or purchase new systems.

5. Related work

Our related work section focuses on the main insights that our research revealed regarding (1) the suitability of the TD metaphor, (2) TD causes, consequences, and visibility, (3) TD communication with business stakeholders and developers, (4) establishing a TDM process, and (5) vintage/legacy systems.

5.1. Technical debt metaphor

On the topic of the TD metaphor, Klaus Schmid (Schmid, 2013) identified the "analogy breaking points" and advised not to over-extend the metaphor's applicability to avoid side-tracking research. Stochel et al. (2020) discussed the ambiguity of the term TD. They attributed this ambiguity to the metaphorical origin of the term, which allows for individual interpretation. Our work supports these perceptions. Most practitioners do not understand the subtleties of the metaphor. Moreover, most of them also do not see the need to use it more intensively.

5.2. Technical debt causes, consequences, and their visibility

Overall, the TD causes and consequence types that we found are quite similar to the findings from the InsightTD project (Rios et al., 2020; Ramač et al., 2021). However, our list is not as extensive on the topic of technical/development type causes and consequences and focuses more on other areas, e.g., business-related causes.

The relationship between TD causes, items, and consequences has been described in various ways by researchers so far as presented in Section 1.4. However, as far as we know, we are the first to divide TD causes into visible and invisible. Additionally, the distinction between consequences and symptoms has been interpreted ambiguously, so far.

Various researchers have noticed "vicious cycles", where the existence of one TD item was the trigger behind incurring growing amounts of debt, e.g., (Besker et al., 2018). This problem has been explored in detail by Martini and Bosch (2015) in the context of ATD, where they explain how ATD growth can be particularly contagious, and thus dangerous.

In our work, we went a step further in describing the relationships between TD causes, consequences, and items on a conceptual level. Not only did we distinguish between outer and inner TD vicious cycles, but we also defined chains of causes and consequences that describe how various causes/consequences impact each other.

5.3. Technical debt communication

Various studies suggest that communication about TD is an important factor in TDM. Verdecchia et al. (2021) describe communication as one of the core categories influencing ATD. Tom et al. (2013) state that poor communication is a cause of inadvertent TD and propose a framework that may help development teams communicate with their managers about TD.

Besker et al. (2022) show how managers communicating the need to keep TD levels low to their development teams can significantly impact the amount of TD in a system. Wiese et al. (2022) show how an established TDM process can make discussions about TD within development teams more rational. This indicates that our idea of a TDM process being the source of better communication with and within development teams might be worth exploring further.

A way of improving communication with business stakeholders is involving them in the TDM process, as Reboucas De Almeida et al. (2018), Reboucas De Almeida (2019) achieved by including both business and technical stakeholders in business-driven TD prioritization. As shown by Kruchten et al. (2012), TD effects may be visible (symptoms) and invisible (consequences) for business stakeholders, which impacts communication. Our V4CTD model enables better communication by explaining communication issues through the concept of visibility.

5.4. Establishing a technical debt management process

There are two recent tertiary studies on TDM by Rios et al. (2018) and by Junior and Travassos (2022) that give an overview of the TDM research field. They both found that TDM is viewed as a set of TD activities and their supporting strategies and tools. Both did not find any research on the systematic establishment of a TDM process. Both also agreed that a holistic approach to TDM, including various TD activities and various TD types, is missing, which makes it hard for practitioners to profit from the research results. This coincides with our finding that IT managers seem to lack suitable guidelines on how to establish TDM.

A general guideline for establishing TDM was presented in the book of Kruchten et al. (2019), particularly, in the "What can you do today?" sections. Yet, the feasibility of this guideline is not evaluated and may need to be improved.

Some case studies provide information on establishing a TDM process. For example, Ramasubbu et al. provided a case study that integrated TD management into quality management steps (Ramasubbu and Kemerer, 2019). Wiese et al. (2022) presented a framework integrating TDM into project management. However, these and other case studies do not provide a guideline for other companies and did not evaluate their work in other companies, e.g. in different work environments.

Additionally, Rios et al. (2018) identified four TDM macro activities, i.e., Prevention, Identification, Monitoring, and Payment. While the idea of macro activities is similar to our clusters of interrelated TD activities, the clusters themselves are different. While Rios et al.'s study's clustering results from an academic literature review, our clustering shows the perception of IT managers in practice.

5.5. Technical debt and vintage/legacy systems

Regarding the relationship between TD and legacy systems, Boss et al. (2016) stated in 2016 that no work on handling architectural technical debt in legacy systems existed and showed a way to deal with these problems using architectural checks. Gupta et al. gave an example of how the amount of TD in a legacy code base was decreased by using agile practices (Gupta et al., 2017). Holvitie et al. stated that over 50% of TD could come from "Legacy from an earlier team/individual working" (Holvitie et al., 2018), which means that dealing with these old "inherited" systems should be a major topic when dealing with TD. In their case study, Lenarduzzi et al. (2020) showed that migrating from a legacy monolithic architecture to a microservice-based one, may cause an initial spike of TD, but does reduce TD in the long run.

Overall, while research on legacy/vintage systems in the context of TD has been done, it has not been very extensive. Our study shows that this is an important topic that should be explored in the future, particularly, researching ways to preserve or adapt parts of those systems.

6. Threats to validity

We present threats to validity based on the guidelines provided by Wohlin et al. (2012).

Construct validity. The selection of participants and interview questions have a great impact on the validity of the study. We used purposeful sampling and achieved code saturation to ascertain that we provided relevant and exhaustive results. Regarding the participants, we overcame the survivorship bias by interviewing former managers and used two iterations to vary the participants in iteration 2 even more. Still, the IT managers already suffering from TD may have been more willing to participate in the study than managers that did not suffer from TD.

Regarding the interview questions, we asked for the most impactful TD item for a detailed analysis. Therefore, the study may be biased towards the impact of ATD on the systems and companies since ATD items were usually considered to be the most impactful.

Internal validity. Internal Validity describes the threats of drawing wrong conclusions about causal relationships between treatment and outcome. Therefore, internal validity mainly refers to quantitative studies such as experiments. In an interview study, there is the possibility of drawing wrong conclusions from the interview transcripts, e.g., by misunderstanding the participants. To make sure that the coding of the interviews did not lead to such misconceptions, the coding was done by two researchers. Additionally, we were able to contact and ask the participants for clarification if some of their explanations could be misunderstood. A strict and communicative process, as described in Section 2, further minimized these internal risks to validity.

External validity. External validity threats refer to the ability to generalize the result. However, the goal of a qualitative study, such as ours, is to create new theories and not generalize the results. The generalizability could be improved by further studies validating the results, e.g., by surveying a representative group of IT managers and confronting them with this study's results.

Conclusion validity. The visibility categorization shown in our V4CTD model is based on interviews with IT managers but provides testimonies about business stakeholders' knowledge of TD. An additional study with business stakeholders, where they evaluate our V4CTD model in practice, should, therefore, follow this study. Moreover, the distinction between visible and invisible causes and consequences highly depends on the business stakeholders' knowledge of the IT domain. Therefore, the distinction might differ from company to company and can never be considered absolute. However, the contribution of this study is that we made the distinction obvious and gave practitioners a starting point for their own 3-minute slide. The focus group comprised team leaders from the Business Intelligence group of a single company. This may cause their insights to be limited to one company and this IT domain. However, since those IT managers' IT domain was underrepresented during the interviews, we were able to test the transferability of our findings by consulting them with this group. Still, the results must be evaluated carefully as they might be biased.

7. Conclusion

We interviewed 16 IT managers and had one three-person focus group discussion. We identified the IT managers' knowledge, their three main concerns regarding TD(M), and the potential improvements they perceived. We compared the results of the study to the current body of knowledge to identify knowledge gaps in research and new insights.

- We uncovered that the original metaphor may not be helpful when conveying the meaning of TD. The financial metaphor is rather used as an an ambiguous informal term and is often not understood in detail.
- We identified causes and consequences on different visibility levels for business stakeholders, IT managers, and development teams. These different levels may be the cause of further communication issues.
- We created the V4CTD model to include chains and (vicious) cycles of causes and consequences as an extension of the TD conceptual model.
- We uncovered the duality of the IT managers' problems in communicating with developers: the "developers crying wolf" and the "it works" developer. Both of these issues can be addressed by establishing a TDM process.
- We observed the impact of the status quo bias on establishing a TDM process and we identified three sub-concerns that need to be addressed to overcome this bias. Two of these sub-concerns can be addressed by establishing a basic TDM process as a starting point. There exists a cyclic dependency between the concerns regarding the TDM's establishment and the TDM's establishment itself, which has to be interrupted by "just starting".
- We displayed the value of old systems and called them vintage systems. These systems were notably mentioned by top managers and occurred mostly in old established companies. Though it is a major issue in practice, research in this field seems to be lacking.

For practitioners, we propose the use of our V4CTD model to overcome miscommunication about TD. Furthermore, we provide insights into the status quo bias and the before-mentioned cyclic dependency. This could lead to more IT managers agreeing to adopt a TDM process.

For researchers, the V4CTD model expands the conceptual model of TD and we provided a sub-structure of interrelated TD activities. Additionally, we identified the need for methods on how to establish a TDM process in a feasible and sustainable way. Finally, we pose the (research) question if and in which cases it might be an option to keep and adapt vintages systems.

7.1. Future work

We suggest evaluating the feasibility of using our V4CTD-model-based three-minute slides in practice and validating the list of visible and invisible causes and consequences by business stakeholders. Additionally, the possible use of other metaphors may be researched further. Moreover, future research on TDM should focus on the establishment of TDM processes and how to overcome the status quo bias. Since we did not analyze the potential improvements regarding vintage systems, we could not provide any aid for practitioners on this topic. Therefore, this might be an additional problem worth researching.

CRediT authorship contribution statement

Marion Wiese: Conceptualization, Methodology, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Project administration. **Klara Borowa:** Conceptualization, Methodology, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data is available on Zenodo as referenced in the manuscript.

Acknowledgment

We thank the participants of our research for their participation, their trust, and the insights they provided.

References

Aesop, 2002. In: Gibbs, L. (Ed.), Aesop's Fables. Oxford University Press.
Ahmadjee, S., Mera-Gomez, C., Bahsoon, R., 2021. Assessing smart contracts security technical debts. In: Proceedings - 2021 IEEE/ACM International Conference on Technical Debt, No. March 2021. TechDebt 2021, pp. 6–15. http://dx.doi.org/10.1109/TechDebt52882.2021.00010.
Anon, 2022a. InsighTD project – InsighTD project. URL http://www.td-survey.com/.
Anon, 2022b. MAXQDA | All-in-one tool for qualitative data analysis & mixed methods - MAXQDA. URL https://www.maxqda.com/.
Anon, 2022c. Legacy code rocks!. URL https://www.legacycode.rocks/.
Anon, 2022d. Jira | Issue & project tracking software | Atlassian. URL https://www.atlassian.com/software/jira.
Anon, 2022e. Trello | Manage your team's projects from anywhere. URL https://trello.com/.
Anon, 2022f. SonarQube. URL https://www.sonarqube.org/downloads/.
Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C., 2016. Managing technical debt in software engineering. Dagstuhl Rep. 6 (4), 110–138. http://dx.doi.org/10.4230/DagRep.6.4.110.
Baltes, S., Ralph, P., 2022. Sampling in software engineering research: a critical review and guidelines. Empir. Softw. Eng. 27 (4), http://dx.doi.org/10.1007/s10664-021-10072-8.
Bennett, K., 1995. Legacy sysfems: Coping with success. IEEE Softw. 12 (1), 19–23. http://dx.doi.org/10.1109/52.363157.
Besker, T., Bosch, J., Martini, A., Bosch, J., 2018. Technical debt cripples software developer productivity: A longitudinal study on developers' daily software development work. In: Proceedings - International Conference on Software Engineering. ACM, pp. 105–114. http://dx.doi.org/10.1145/3194164.3194178.
Besker, T., Martini, A., Bosch, J., 2022. The use of incentives to promote technical debt management. Inf. Softw. Technol. 142, http://dx.doi.org/10.1016/j.infsof.2021.106740.

Boss, B., Tischer, C., Krishnan, S., Nutakki, A., Gopinath, V., 2016. Setting up architectural SW health builds in a new product line generation. In: Proccedings of the 10th European Conference on Software Architecture Workshops. ECSAW '16, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/2993412.3003392.

Campbell, J.L., Quincy, C., Osserman, J., Pedersen, O.K., 2013. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. Sociol. Methods Res. 42 (3), 294–320. http://dx.doi.org/10.1177/0049124113500475.

Corbin, J., Strauss, A., 2014. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, fourth ed. Sage publications.

Cunningham, W., 1992. The WyCash portfolio management system. In: Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, Vol. Part F1296, No. 2. OOPSLA, pp. 29–30. http://dx.doi.org/10.1145/157710.157715.

Ernst, N., Kazman, R., Delange, J., 2021. Technical Debt in Practice. The MIT Press Cambridge, Massachusetts.

Garrison, D.R., Cleveland-Innes, M., Koole, M., Kappelman, J., 2006. Revisiting methodological issues in transcript analysis: Negotiated coding and reliability. Internet High. Educ. 9 (1), 1–8. http://dx.doi.org/10.1016/j.iheduc.2005.11.001.

Gomes, F.G., Dos Santos, E.P., Mendes, T.S., De Mendonça Neto, M.G., Silva Freire, E.S., Spínola, R.O., 2022. Investigating the point of view of project management practitioners on technical debt - A preliminary study on stack exchange. In: Proceedings - 2022 IEEE/ACM International Conference on Technical Debt, Vol. 1, No. 1. TechDebt 2022, ACM, pp. 31–40. http://dx.doi.org/10.1145/3524843.3528095.

Gupta, R.K., Manikreddy, P., Arya, K.C., 2017. Pragmatic scrum transformation: Challenges, practices & impacts during the journey a case study in a multi-location legacy software product development team. In: Proceedings of the 10th Innovations in Software Engineering Conference. ISEC '17, Association for Computing Machinery, New York, NY, USA, pp. 147–156. http://dx.doi.org/10.1145/3021460.3021478.

Holvitie, J., Licorish, S.A., Spínola, R.O., Hyrynsalmi, S., MacDonell, S.G., Mendes, T.S., Buchan, J., Leppänen, V., 2018. Technical debt and agile software development practices and processes: An industry practitioner survey. Inf. Softw. Technol. 96, 141–160. http://dx.doi.org/10.1016/j.infsof.2017.11.015.

Jain, S., Babar, M.A., Fernandez, J., 2013. Conducting empirical studies in industry: Balancing rigor and relevance. In: 2013 1st International Workshop on Conducting Empirical Studies in Industry, CESI 2013 - Proceedings. IEEE, pp. 9–14. http://dx.doi.org/10.1109/CESI.2013.6618463.

Junior, H.J., Travassos, G.H., 2022. Consolidating a common perspective on technical debt and its management through a tertiary study. Inf. Softw. Technol. 149, http://dx.doi.org/10.1016/j.infsof.2022.106964.

Kahneman, D., Knetsch, J.L., Thaler, R.H., 1991. Anomalies: The endowment effect, loss aversion, and status quo bias. J. Econ. Perspect. 5 (1), 193–206.

Khadka, R., Batlajery, B.V., Saeidi, A.M., Jansen, S., Hage, J., 2014. How do professionals perceive legacy systems and software modernization? In: Proceedings - International Conference on Software Engineering, No. 1. pp. 36–47. http://dx.doi.org/10.1145/2568225.2568318.

Kruchten, P., Nord, R., Ozkaya, I., 2019. Managing Technical Debt: Reducing Friction in Software Development. Addison-Wesley Professional.

Kruchten, P., Nord, R.L., Ozkaya, I., Visser, J., 2012. Technical debt in software development. ACM SIGSOFT Softw. Eng. Notes 37 (5), 36–38. http://dx.doi.org/10.1145/2347696.2347698.

Lenarduzzi, V., Lomio, F., Saarimäki, N., Taibi, D., 2020. Does migrating a monolithic system to microservices decrease the technical debt? J. Syst. Softw. 169, http://dx.doi.org/10.1016/j.jss.2020.110710.

Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. J. Syst. Softw. 101, 193–220. http://dx.doi.org/10.1016/j.jss.2014.12.027.

Martini, A., Besker, T., Bosch, J., 2018. Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. Sci. Comput. Program. 163, 42–61. http://dx.doi.org/10.1016/j.scico.2018.03.007.

Martini, A., Bosch, J., 2015. The danger of architectural technical debt: Contagious debt and vicious circles. In: Proceedings - 12th Working IEEE/IFIP Conference on Software Architecture. WICSA2015, pp. 1–10. http://dx.doi.org/10.1109/WICSA.2015.31.

Pérez, B., Castellanos, C., Correal, D., Rios, N., Freire, S., Spínola, R., Seaman, C., Izurieta, C., 2021. Technical debt payment and prevention through the lenses of software architects. Inf. Softw. Technol. 140, http://dx.doi.org/10.1016/j.infsof.2021.106692.

Ramač, R., Mandić, V., Taušan, N., Rios, N., Freire, S., Pérez, B., Castellanos, C., Correal, D., Pacheco, A., Lopez, G., Izurieta, C., Seaman, C., Spinola, R., 2021. Prevalence, common causes and effects of technical debt: Results from a family of surveys with the IT industry. J. Syst. Softw. http://dx.doi.org/10.1016/j.jss.2021.111114.

Ramasubbu, N., Kemerer, C.F., 2019. Integrating technical debt management and software quality management processes: A framework and field test. Test Eng. Manag. 45 (7–8), http://dx.doi.org/10.1145/3180155.3182529.

Reboucas De Almeida, R., 2019. Business-driven technical debt prioritization. In: Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution. ICSME 2019, pp. 605–609. http://dx.doi.org/10.1109/ICSME.2019.00096.

Reboucas De Almeida, R., Kulesza, U., Treude, C., Cavalcanti Feitosa, D., Lima, A.H.G., 2018. Aligning technical debt prioritization with business objectives: A multiple-case study. In: Proceedings - 2018 IEEE International Conference on Software Maintenance and Evolution. ICSME 2018, pp. 655–664. http://dx.doi.org/10.1109/ICSME.2018.00075.

Rios, N., de Mendonça Neto, M.G., Spínola, R.O., 2018. A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. In: Information and Software Technology, Vol. 102. pp. 117–145. http://dx.doi.org/10.1016/j.infsof.2018.05.010.

Rios, N., Oliveira Spínola, R., de Mendonça Neto, M.G., Seaman, C., 2019. Supporting analysis of technical debt causes and effects with cross-company probabilistic cause-effect diagrams. In: 2019 IEEE/ACM International Conference on Technical Debt. TechDebt, pp. 3–12. http://dx.doi.org/10.1109/TechDebt.2019.00009.

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. Empir. Softw. Eng. 25 (5), 3216–3287. http://dx.doi.org/10.1007/s10664-020-09832-9.

Rocha, J.C., Zapalowski, V., Nunes, I., 2017. Understanding technical debt at the code level from the perspective of software developers. In: ACM International Conference Proceeding Series, No. i. pp. 64–73. http://dx.doi.org/10.1145/3131151.3131164.

Runeson, P., Höst, M., Rainer, A., Regnell, B., 2012. Case study research in software engineering: Guidelines and examples. In: Case Study Research in Software Engineering: Guidelines and Examples. http://dx.doi.org/10.1002/9781118181034.

Saldaña, J., 2013. the Coding Manual for Qualitative Researchers, second ed. SAGE publications, London, England, http://dx.doi.org/10.1108/qrom-08-2016-1408.

Saunders, B., Sim, J., Kingstone, T., Baker, S., Waterfield, J., Bartlam, B., Burroughs, H., Jinks, C., 2018. Saturation in qualitative research: exploring its conceptualization and operationalization. Qual. Quant. 52 (4), 1893–1907. http://dx.doi.org/10.1007/s11135-017-0574-8.

Schmid, K., 2013. On the limits of the technical debt metaphor: Some guidance on going beyond. In: Fourth Workshop on Managing Technical Debt, Workshop At the International Conference on Software Engineering. IEEE, pp. 63–66. http://dx.doi.org/10.1109/MTD.2013.6608681.

Soliman, M., Avgerioua, P., Lia, Y., Avgeriou, P., Li, Y., 2021. Architectural design decisions that incur technical debt — An industrial case study. Inf. Softw. Technol. 139, http://dx.doi.org/10.1016/j.infsof.2021.106669.

Stochel, M.G., Cholda, P., Wawrowski, M.R., 2020. On coherence in technical debt research : Awareness of the risks stemming from the metaphorical origin and relevant remediation strategies. In: Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications. SEAA2020, pp. 367–375. http://dx.doi.org/10.1109/SEAA51224.2020.00067.

Stol, K.J., Fitzgerald, B., 2015. Theory-oriented software engineering. Sci. Comput. Program. 101, 79–98. http://dx.doi.org/10.1016/j.scico.2014.11.010.

Tom, E., Aurum, A., Vidgen, R., 2013. An exploration of technical debt. J. Syst. Softw. 86 (6), 1498–1516. http://dx.doi.org/10.1016/j.jss.2012.12.052.

Verdecchia, R., Kruchten, P., Lago, P., Malavolta, I., 2021. Building and evaluating a theory of architectural technical debt in software-intensive systems. J. Syst. Softw. 176, http://dx.doi.org/10.1016/j.jss.2021.110925.

Vogel-Heuser, B., Bi, F., 2021. Interdisciplinary effects of technical debt in companies with mechatronic products — a qualitative study. J. Syst. Softw. 171, http://dx.doi.org/10.1016/j.jss.2020.110809.

Wiese, M., Borowa, K., 2022. Additional material for IT managers' perspective on technical debt management. http://dx.doi.org/10.5281/zenodo.7803013.

Wiese, M., Rachow, P., Riebisch, M., Schwarze, J., 2022. Preventing technical debt with the TAP framework for technical debt aware management. Inf. Softw. Technol. http://dx.doi.org/10.1016/j.infsof.2022.106926.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. Experimentation in software engineering. In: Experimentation in Software Engineering. Springer Science & Business Media, pp. 1–236.

Yli-Huumo, J., Maglyas, A., Smolander, K., 2016. How do software development teams manage technical debt? – An empirical study. J. Syst. Softw. 120, 195–218. http://dx.doi.org/10.1016/j.jss.2016.05.018.

Marion Wiese is a Research Assistant at the Universität Hamburg. She obtained her Diploma in Informatics at Universität Hamburg in 2001. Her main research interest is technical debt management. She has published three peer-reviewed papers so far. She lectures Software Architecture, Project Management, and Software Re-Engineering.

From 2001–2020 she worked at Gruner+Jahr GmbH as a software architect and senior developer. She was part of many different projects, like the optimization of a distributed environment's integration architecture, the definition of her company's reference architecture for business intelligence applications, and the development and project management of a subscription management system for magazines.

Klara Borowa is a Research Assistant at the Warsaw University of Technology. She obtained her Master of Science degree in Computer Science at Warsaw University of Technology. Her main research interest is the influence of human factors on software engineering and software architecture. She authored papers published at various venues such as ICSA (International Conference on Software Architecture), ECSA (European Conference on Software Architecture) and IEEE Software. Previously she worked in the software industry at companies such as Hewlett Packard Enterprise, DXC Technology, and Atos.