The use of incentives to promote technical debt management

Citation for the original published paper (version of record):
Besker, T., Martini, A., Bosch, J. (2022). The use of incentives to promote technical debt management. Information and Software Technology, 142.
http://dx.doi.org/10.1016/j.infsof.2021.106740

Information and Software Technology 142 (2022) 106740

Contents lists available at ScienceDirect

Information and Software Technology

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

The use of incentives to promote technical debt management

Terese Besker [a,*], Antonio Martini [b], Jan Bosch [c]

[a] RISE Research Institutes of Sweden AB, Gothenburg, Sweden
[b] University of Oslo Programming and Software Engineering, Oslo, Norway
[c] Computer Science and Engineering, Software Engineering, Chalmers University of Technology, Gothenburg, Sweden

ARTICLE INFO

Editor: Dr Jeff Carver

Keywords:
Technical debt
Software development
Software incentive programs
Empirical study

ABSTRACT

Context: When developing software, it is vitally important to keep the level of technical debt down since, based on several studies, it has been well established that technical debt can lower the development productivity, decrease the developers' morale and **compromise** the overall quality of the software, among others. However, even if researchers and practitioners working in today's software development industry are quite familiar with the concept of technical debt and its related negative consequences, there has been no empirical research focusing specifically on how software managers actively communicate and manage the need to keep the level of technical debt as low as possible.

Objective: This study aims to understand how software companies give incentives to manage technical debt. This is carried out by exploring how companies encourage and reward practitioners for actively keeping the level of technical debt down add whether the companies use any *forcing* or *penalising* initiatives when managing technical debt.

Method: As a first step, this paper reports the results of both an online survey providing quantitative data from 258 participants and interviews with 32 software practitioners. As a second step, this study sets out to specifically provide a detailed assessment of additional and in-depth analysis of technical debt management strategies based on an encouraging mindset and attitude from both managers and technical roles to understand *how, when and by whom* such strategies are adopted in practice.

Results: Our findings show that having a technical debt management strategy (specially based on encouragement) can significantly impact the amount of technical debt related to the software.

Conclusion: The result indicates that there is considerable unfulfilled potential to influence how software practitioners can further limit and reduce technical debt by adopting a strategy based explicitly on an encouraging mindset from managers where they also specifically dedicate time and resources for technical debt remediation activities.

1. Introduction

When developing software, it is vitally important to keep the level of technical debt (TD) down since it is well established from several previous studies that TD can, for example, lower development productivity [1], decrease the developers' morale [2] and compromise the overall software quality [3] and even lead to a crisis point when a huge, costly refactoring or replacement of the whole software needs to be undertaken [4].

The TD metaphor was first introduced by Ward Cunningham [5] to illustrate the need to recognise the potential long-term negative effects of immature code that is sub-optimally implemented during the software development lifecycle. This debt must be repaid with interest over the long term [6].

Even if the concept of TD and its negative consequences is quite well known to software engineering (SE) practitioners today, there is always a risk that TD remediation tasks are down-prioritised or neglected by practitioners since today's software practitioners face increased pressure from management to reduce the development time and, thereby, to reduce the costs of development [7].

On the other hand, at the same time, it is important to deliver high-quality software with as little TD as possible. There is a balancing act that becomes particularly demanding; to implement and deliver the software as quickly as possible while also spending time and effort

* Corresponding author.
E-mail addresses: Terese.Besker@ri.se (T. Besker), antonima@ifi.uio.no (A. Martini), Jan.Bosch@chalmers.se (J. Bosch).

https://doi.org/10.1016/j.infsof.2021.106740
Received 29 December 2020; Received in revised form 23 September 2021; Accepted 28 September 2021
Available online 7 October 2021

avoiding the introduction of TD in the first place, as well as conducting TD refactoring activities for software that has already been implemented.

Like other professionals, software engineers' work outcomes, attitudes, and work behaviors are influenced by their company's corporate culture and the managers' mindset [8]. This means that managers can have an outsized impact on the overall software development process by adopting different management strategies and using techniques for controlling and directing software engineers to achieve predetermined goals.

In recent years, the use of different strategies in behavioral interventions has become more prevalent [9]. In our literature review (see Section 2.2), this study initially identifies four different strategies that managers can adopt to impact how practitioners work with TD. Besides *encouraging* employees by, for example, introducing training programs that focus on raising awareness and enhancing knowledge about specific desired behavior, there are also other strategies managers can implement to impact [10] and motivate [11,12] their employees. In general, one mechanism managers use to impact practitioners' work is an incentive program, where a specific behavior is recognised and *rewarded* [13,14]. To have the opposite effect, managers can also use disincentive programs to *penalize* an undesired or destructive behaviour [15]. Furthermore, managers can similarly implement explicitly *forced* requirements and rules, with all employees concerned expected to fulfil and adapt to these in order to continue their work or, for example, to deploy their implementations and continue developing new tasks [16].

However, to the best of our knowledge, this is the first study to investigate empirically how common and important different management strategies are when specifically managing TD in today's software development industry.

This study is carried out in two steps. Firstly, we will study four different TD management incentive strategies to manage TD (addressing RQ 1-4). Secondly, based on the findings in the first step, we provide a detailed assessment of additional in-depth analysis of one of these strategies (encouragement) in order to understand *how, when and by whom* such a strategy is adopted in practice (addressing RQ 5-7).

In particular, this study examines the following seven main research questions:

RQ1: How common is an *encouraging* attitude to keep the level of TD down?

RQ1.1: Do software engineering practitioners perceive this TD management strategy as an effective or desirable strategy?

RQ2: How common are *rewarding* incentives to keep the level of TD down?

RQ2.1: Do software engineering practitioners perceive this TD management strategy as an effective or desirable strategy?

RQ3: How common is it to use a *forcing* mechanism to keep the level of TD down?

RQ3.1: Do software engineering practitioners perceive this TD management strategy as an effective or desirable strategy?

RQ4: How common are *penalising* disincentives to keep the level of TD down?

RQ4.1: Do software engineering practitioners perceive this TD management strategy as an effective or desirable strategy?

RQ5: What specific TD management activities/tasks are encouraged and who encourages these activities?

RQ6: In what situations or under what circumstances are practitioners encouraged to address TD?

RQ7: How are practitioners encouraged to address TD?

This paper reports the results of two sets of surveys: The first is an online survey providing quantitative data from 258 respondents, and the second survey provides data from 72 respondents. The result is also based on qualitative data from interviews with initially 32 software practitioners from seven software companies, followed by yet another round of four interviews. All surveys used are included in Appendix A.

To the best of our knowledge, no known empirical research has focused on exploring the relationships between TD and different management strategies. The contribution to this subject in this paper is fourfold: Firstly, we show how common it is to use each of the investigated strategies within today's software industry. Secondly, our result shows that a TD management strategy can significantly impact the amount of TD in the software. Thirdly, when surveying how commonly it is to use different TD management strategies, we found that only the encouraging strategy is, to some extent, adopted in today's software industry. Lastly, our result clearly shows that there is a misalignment between *how* and *when* managers perceive they encourage the development teams to address TD in comparison to how and in what situations or under what circumstances the teams perceive being encouraged by their managers.

Taken together, these findings provide valuable insights into the role that management has on the way practitioners address TD during their software development work.

The rest of this paper is organised as follows. Section 2 introduces the background and the related work. Section 3 describes the research methods in detail. Section 4 presents the research results. Section 5 discusses the findings, Section 6 presents threats to the study's validity and Section 7 concludes the study.

Since encouragement was found to be the most important activity used in practice by practitioners (see Section 4.2), this strategy encompasses a more detailed perspective where further aspects are covered related to *how, when*, and *by whom* encouragement is carried out in practice.

2. Background and related work

This section presents related work concerning incentive and disincentive programmes in today's software engineering field, followed by the different management strategies, as illustrated in the conceptual framework in Fig. 1.

2.1. Incentives and disincentive programmes in SE

An incentive programme addresses a planned activity designed to motivate employees (individuals or teams) to achieve specified and predetermined organisational goals or objectives within a specific timeframe. At the same time, a disincentive programme is the antonym of the incentive programme and discourages employees (individuals or teams) from performing specific activities [17].

Commonly, software development projects are measured using financial indicators. Typically, the incentive programme aims to give bonuses to managers who run their projects with a high profit margin or within the budget timeframe [18]. There is no research to date on how other software engineering roles, such as those of developers, testers, and architects, are included in incentive or disincentive programmes in general and, more specifically, in TD management.

2.2. Conceptual framework

Organisations can use different management strategies to influence employees' working behaviours. Initially, as a rational for understanding more about these different strategies, we search for research publications addressing different strategies that managers can use for influencing practitioners' working behaviours and attitudes in both SE-related research sources and also in other disciplines using a conceptual review approach [19],[20].

By using a conceptual review approach, where the design strategy was inspired by Hulland [20], our search was not limited to searches by strict terms/words as described by Ayala [19]. Thus, we were able to conduct a broader search for different strategies on how to manage and impact practitioners seen from a manager's point of view. This approach was useful when constructing our conceptual framework: *"In building conceptual families, a typology begins to form for the retrieved sources*

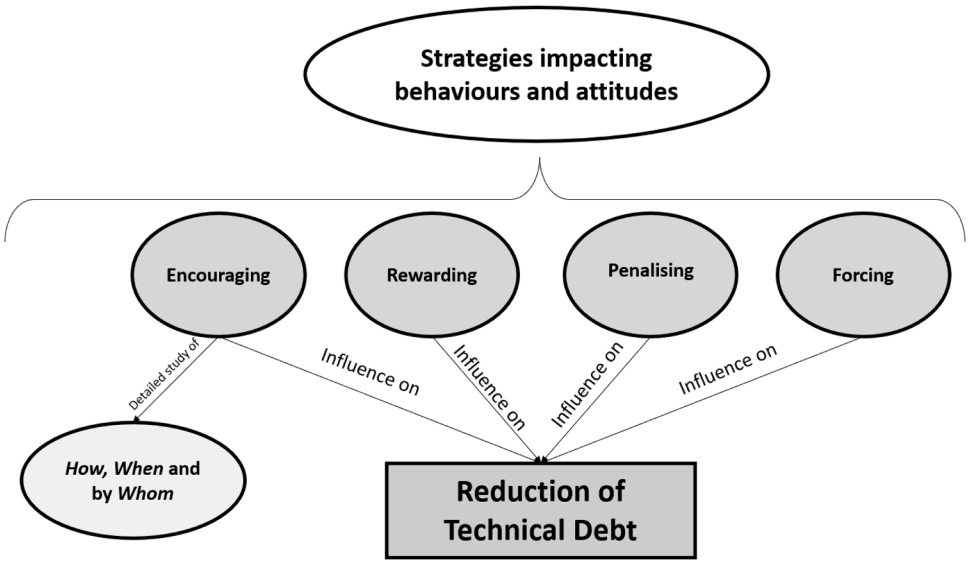

Fig. 1. Conceptual framework.

grouped according to the social and historical contexts that they stem from. This is very helpful in theoretically framing the research problem" [19]. However, it is important to acknowledge that since we did not fully adopt the strategy suggested by Hulland [20] using a fully systematic search approach, there may potentially be more strategies that unwittingly were left out of the framework.

Based on this conceptual review's outcome, we have depicted a conceptual model presented in Fig.1 and formulated the research questions based on this framework. The framework illustrates four main different management strategies; a) Encouraging, b) Rewarding, c) Forcing, d) Penalising.

The conceptual framework with its four main strategies is proposed under the rationale that these four strategies are potentially important and can influence the reduction of TD. Potentially, they are being used by managers to manage the amount of TD during the software development work. The sources of each of the identified strategies are presented in next sections.

2.2.1. Encouraging activities

Encouraging employees is an important part of being a leader where the leader highlights and complements specific desired actions and where the leader also provides constructive criticism if needed. Managers' behaviours provide an important message to employees, meaning, for example, that a high level of creativity and innovation result from managerial behaviours [21] where the relationship between employees and their managers has a significant bearing on employees' work-related attitudes and behaviours [10]. By default, simply encouraging employees does not include any direct rewards.

2.2.2. Rewarding incentive

Several studies show that reward and recognition programmes can positively influence motivation, performance, and interest within an organisation [13,14]. The overall goal with reward and/or recognition programmes is to foster teamwork, boost employee loyalty, and ultimately facilitate the development of a desired culture that rewards a specific behaviour [13]. The practitioners (individuals or teams) who fulfil the goals receive a predefined reward. A reward programme can, for example, recognise developers who adopt suggested techniques, and thereby the reward incentive gives a significant boost to those who deploy best practices, where the achievement, for example, can be rewarded by a badge [22], by a gift card or a monetary reward.

2.2.3. Forcing mechanisms

The strategy based on forcing mechanisms refers to mandatory rules and requirements that need to be fulfilled and followed by the practitioners to demonstrate adherence to methodologies, rules, regulations, guidelines or best practices [16]. This could be exemplified by a situation where mandatory rules and requirements are not met. Hence, practitioners are forced to go back and alter the software before being allowed to continue with, for example, adding additional features or deploying the software. Examples of commonly adopted rules and requirements from an SE perspective include:

- Not allowing any bugs in the software
- Requiring that the software be thoroughly tested before deployment
- Ensure code is fully reviewed
- Ensuring the code follows the coding standards

2.2.4. Penalising disincentive

Organisational penalties or punishments are part of a pervasive phenomenon in many companies and organisations [23] that yield penalties for undesired behaviour and are also part of a disincentive strategy. Penalisation refers to when managers apply a negative consequence or the removal of a positive consequence following an employee's undesirable behaviour, intending to decrease the frequency of that behaviuor [15]. According to Wang and Zhang [23], some software development organisations have adopted punishment measures in an attempt to improve software developers' performance, reduce software defects and thereby ensure software quality. The result of penalty mechanisms shows that while these help to reduce software defects in daily coding activities, they fail to achieve programmers' maximum work potential. In their study [23], penalty rules were introduced when software developers were tracked by submitting unsuccessful submissions, which caused monetary fines for the individual developer.

Since there is a current gap in research addressing how different TD management strategies impact upon how practitioners work with TD, this study built on research conducted in other domains and examines the current state of different management strategies in the SE field. Our work is, therefore, different from the studies mentioned above in several aspects:

(a) We provide results derived from data from a real software development environment rather than discussion without empirical evidence as support
(b) We combine both qualitative and quantitative methods

(c) Our study investigates four different management strategies
(d) Our investigation primarily focuses on TD.

2.3. TD management activities

TD management facilitates decision-making about the need to remove or avoid a TD item and the most appropriate time to do so [24]. There are several different TD management activities, which may significantly impact the amount of TD within a software system. Based on our previous studies addressing this topic, we have identified below the activities that may impact the amount of TD in a system.

Since TD has a significant negative impact on software development work from several different perspectives, our previous research [25] show that it is essential to actively prevent the introduction of TD into the software in the first place and to iteratively and continuously conduct TD tasks when it has already been introduced into the software. This means that targeting and encouraging software practitioners to perform such activities (e.g., avoiding and removing TD) may significantly impact reducing the harmful effects of TD [26].

However, to remediate or refactor TD, its identified elements first need to be tracked and prioritised, preferably using an official backlog. Our previous research [7] indicates that when practitioners use so-called “shadow backlogs", TD items may potentially be overlooked during the prioritisation process, the result being that TD items will remain in the software.

Moreover, there are situations where deliberately taking on TD may be a strategically sound move since such conscious decisions sometimes may increase the ability to cut development time, thereby enable fast feedback from customers and increase revenue [27–29]. Therefore, in addition to investigate strategies to keep the level of TD down, this study also addresses the extent to which management does exactly the opposite by assessing whether teams are encouraged to deliberately take on additional TD.

Besides encouraging the avoidance and the remediation of TD, it is also essential to address *when* TD refactoring activities should occur and whether the practitioners are empowered to make such decisions on their own or if such decisions must be taken together with managers [7]. The activities and situations presented above are used in the survey in step 2 of the study.

2.4. Our previous work

This manuscript was originally and partly published at the Third International Conference on Technical Debt, held jointly with the ICSE [30]. The delta of this manuscript over the prior published paper is based on an additional empirical extension of the previous study where this manuscript includes an in-depth value-added analysis of the results derived from the first study to provide a more detailed and comprehensive understanding and perception of the first sets of results. This extended study also addresses the comments we received from the anonymous reviewers during the first submission.

This manuscript has been extended to include three additional research questions (RQ5-7) where the results of these questions are derived from a totally new set of independent data collection.

The related Research section has been extended to be broader and to more carefully cover additional related research publications to address the additional research question.

The Methodology section is updated to also includes the additional step of the study together with an illustration and description of how the different steps of this study relate to each other.

Furthermore, in both the Result and the Discussion sections of this extended version of the study, several new findings and results have been added and discussed. These additional results highlight the previous results and further strengthen the ability to understand the first set of results, thereby bringing a finer granularity to our understanding of the practice.

3. Methodology

As visualised in Fig. 2, this study used a combination of quantitative and qualitative research approaches. The research design was divided into two main steps, including a total of 11 different phases. As illustrated in the Fig., the first six phases were conducted in step 1, which refers to this publication’s original study [30]. The following phases (7 to 11) were conducted as an extension to that study.

The research approach used in step 1 is characterised by an exploratory approach where several different incentive strategies are studied. This step’s outcome is used as input to the following step 2. This is characterised by a conclusive study approach where this step primarily focuses on one type of incentive programme from step 1.

The following sections describe each step with its phase together with its related research methods.

3.1. Step 1 – The exploratory part of the study

The first step of this study’s *exploratory nature* aims to answer RQ1-RQ4 and is described in the following six sub-sections.

3.1.1. Phase 1 – Contextual analysis and design

The study was first presented and discussed during a workshop with software practitioners from seven software companies within our industrial network. All companies had an extensive range of software development work. The goal of the workshop was to create a research design, and the outcome is the research model shown in Fig. 2, which directed the design, data collection, and analysis of the following phases.

This step of the study adopted a selection of respondents using mainly a *purposive sampling technique* [31] of software professionals. The aim of primarily using a purposive sampling technique was to select relevant and suitable candidates for the study. Altogether, out of the 258 respondents in the survey, 21 respondents came from LinkedIn invitations from software engineering groups and the remaining 247 respondents came from our network of seven of our industrial software partners. All these seven companies had an outspoken strategy and goal of enhancing their TD remediation work and thereby strived to reduce the negative effects of TD. Characteristics of the sample survey is presented in Table 3.

3.1.2. Phase 2 – Quantitative data collection (DC1)

The data collected in this phase was supported by an online web survey designed and hosted by SurveyMonkey. The motivation for using a survey in this part of the study was to reach a high level of generalisation based on a large population of software professionals [32]. According to the guidance provided by Czaja and Blair [33], the first draft of the survey was tested by four industrial practitioners (a developer, manager, project owner, and software architect) and by two Ph.D. candidates to evaluate the understanding of the questions and the use of common terms and expressions [33]. During this evaluation, we also monitored the time needed to complete the survey.

The survey invitations were emailed to the same *seven* companies that participated in Phase 1, all located in Scandinavia, having an extensive range of software development experience. The invitations were also published on software engineering-related *networks on LinkedIn*. The surveys were anonymous and participation in the surveys was voluntary.

The first part of the survey gathered descriptive statistics to summarise the respondents’ backgrounds and their companies.

The second part of the survey included the four survey statements (ST) presented in Table 1 to facilitate quantitative answers for the RQs presented in Section 1 (when fully answering the RQs, we used quantitative data from this phase combined with qualitative data as described in section 3.1.6), and using survey design guidelines provided by Díaz de Rada [34].

For each of the statements, the respondents were asked to indicate

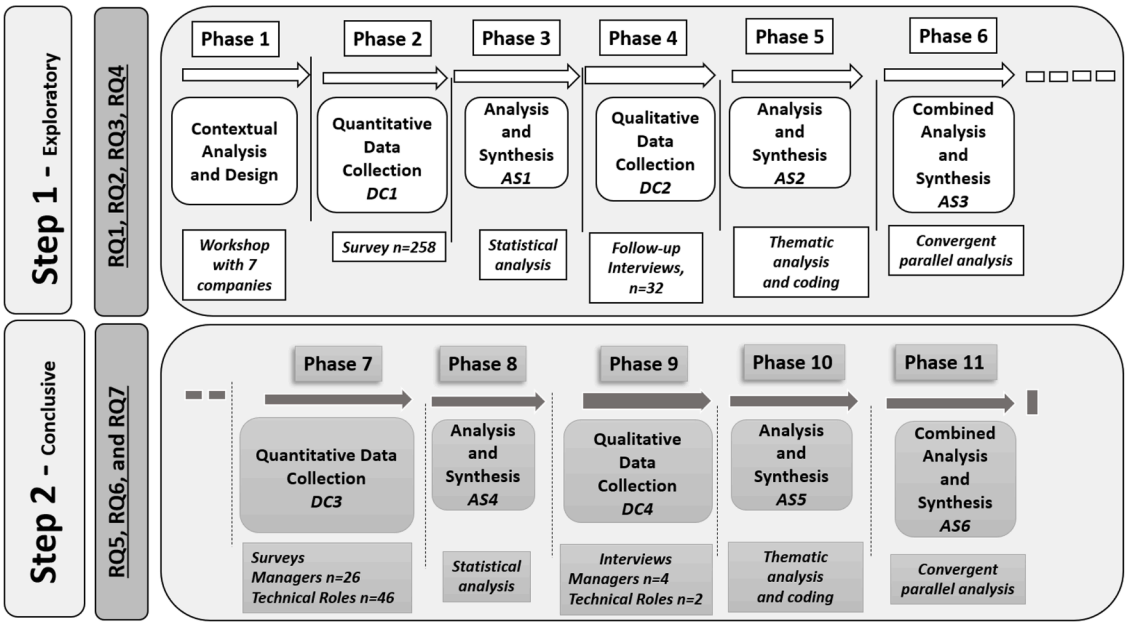

Fig. 2. Research Design.

Table 1
Characteristics of the sample survey – all roles in Step 1.

ID	Statement	Addressing RQ
ST1	Our team is or I am explicitly rewarded if TD is kept down.	2
ST2	Our team is or I am explicitly penalised if TD is not kept down.	4
ST3	Our team is or I am explicitly forced to keep the level of TD down (i.e., to be allowed for deployment)	3
ST4	Our team is or I am explicitly encouraged if TD is kept down.	1

their level of agreement on the 6-point Likert scale; Strongly Agree, Agree, Somewhat Agree, Somewhzat Disagree, Disagree, and Strongly Disagree.

Phase 3 – Analysis and synthesis (AS1)

The survey data was analysed quantitatively, that is, by interpreting the numbers obtained from the answers. The data was analysed using descriptive statistics and graphically visualised using diverging stacked bar charts. The motivation for using diverging stacked is based on the guidelines from Heiberger and Robbins [43] who state that "*We recommend diverging stacked bar charts as the primary graphical display technique for Likert and related scales", also, research conducted by Indratmo et al. [44] conclude that diverging stacked bar charts are easier-to-use and Streit and Gehlenborg [45] state that "Bar charts and box plots are omnipresent in the scientific literature. They are typically used to visualize quantities associated with a set of items.*"

To further quantitatively test our findings, we have also conducted a chi-squared test of independence. This was used especially to test if different roles answering the questionnaire gave significantly different answers, which was relevant for our research questions. In particular, we have analyzed the answers in pairs, always using the answers to the "role" as a variable and the agreement answers to different statements as the other variable. Given the low amount of respondents in some specific roles, the roles were grouped in "managerial" and "technical" to mitigate threats related to running chi-square with too few data points. If the test of independence would show a low p-value, it means that we would have enough evidence to statistically claim that there is a significant difference between the chosen variables. In the specific cases, it would confirm (or not) that different roles had answered differently to the different statements.

3.1.3. Phase 4 – Qualitative data collection (DC2)

In this stage, the second round of data was collected, where 32 software practitioners were focus-group-interviewed. As suggested by Runeson and Höst [35], this study employed the technique of semi-structured interviews, where the questions were planned but not necessarily asked in the same order as they were listed. These interviews were used to obtain detailed information about the interviewees' perceptions and interpretations of the study topics. Examples of interview questions are presented in Table 4.

All interviews were focus-group interviews based on guidelines by Krueger and Casey [36], stating that this method is specifically suitable, serving as a source of follow-up data to assist a prior used data collection method: "*The researchers need the information to help shed light on quantitative data already collected.*"

In total, we interviewed seven companies where each interview included between four to seven interviewees. Altogether, we interviewed 32 experienced software development professionals with roles as architects, developers, product owners, and managers. All interviewees had participated in the previous survey. For confidentiality, interviewees and their companies were anonymised.

All interviewees were asked for recording permission before starting and they all agreed to be recorded and to be anonymously quoted for this paper. Each interview lasted between 105 and 120 minutes and was digitally recorded and transcribed verbatim. Examples of interview questions for each RQ are presented in Appendix A.

Before the interviews started, the previous survey's compiled results were presented to the respondents (using graphical illustrations such as bar diagrams and graphs). This presentation allowed the respondents to relate the interview questions to the results of the survey more easily.

The interview questions were designed to a) increase the understanding of the survey results, b) ensure that the survey questions were understood and interpreted as intended and uniformly, c) confirm the survey results, and d) understand the survey results' implications. The questions were developed to cover the same taxonomies as the previous survey to validate the findings of the survey.

3.1.4. Phase 5 – Analysis and synthesis (AS2)

This stage focused on analysing the data collected in the previous

phase. The data analysis and synthesis were performed using thematic analysis [37]. Thematic analysis is a reliable data analysis method for capturing and reporting themes and the analysis is especially suitable for studying the attitudes and behavior of people [38].

When analysing the qualitative data, the guidelines provided by Braun and Clarke [39] were used to conduct the analysis in a thorough manner.

First, the audio-recorded interviews were transcribed into a written form so we were also able to familiarise ourselves with the data. The second step involved producing initial codes from the data, where we organised the data into suitable groups. Next, we focused on searching for themes by sorting the different codes into potential themes and collecting all the relevant coded data extracts within each identified theme. In this phase of the analysis, a qualitative data analysis (QDA) software package called Atlas.ti was used. For example, the citation "*I think it sounds like the reward would be the best way of keeping the measure of technical debt down*" was coded as "Rewarding."

To ensure that the coding was performed consistently and reliably, two authors of this study synchronised the coding output as suggested by Campbell et al. [40]. The coding process was performed iteratively until reaching a state of saturation (due to the richness of the data and we stopped at the point where no additional codes or categories were identified).

The outcome of this analysis process (where the mapping between the different hierarchical categories and individual codes is presented graphically) is illustrated in Fig. 3.

3.1.5. Phase 6 – Combined analysis and synthesis (AS3)

In the sixth phase of the study, we combined the results we received in the previous quantitative and qualitative data collection and analysis phases.

3.2. Step 2 – The conclusive part of the study

The second step of this study's *conclusive* nature aims to answer RQ5, RQ6, and RQ7. The background of the research conducted in this step is based on the results derived in Step 1, where about 60% of the survey respondents state that they were directly encouraged by managers to keep the level of TD down (see Section 4.2.1).

As a result of this finding, in step 2, therefore, we further specifically investigate a TD management strategy based on explicit encouragement to better understand how and by whom this encouragement is carried out in practice.

In this step of the study, we conducted a single, case study, following the guidelines provided by Runeson and Höst [35]. We consider the case as embedded as multiple units of analysis were studied within the case (different groups of roles). The selected case company is a large technological development supplier with around 385,000 employees operating worldwide. The participating respondents work at several different sites, operate in different countries, and have different managers. This case study company was selected due to its *ongoing initiative addressing TD* during their software development work, making it suitable for studying an incentive strategy based on encouragement. For confidentiality reasons, the company name has been anonymised in this study.

3.2.1. Phase 7 – Quantitative data collection (DC3)

The data in this phase was collected using a similar approach as in phase 2, Step 1 where the quantitative data was collected using the online service provided by Surveymonkey. However, this phase collected quantitative data using two totally new sets of surveys; one for managers and one for technical roles.

The first parts of each of the surveys gathered descriptive statistics to summarise the backgrounds of the respondents. This data is presented in detail in Appendix A. Altogether, 26 managers and 46 technical roles participated in the surveys.

In these two surveys, all respondents were asked to indicate their agreement level on a 4-point Likert scale (Strongly Agree, Agree, Disagree, and Strongly Disagree). The statements assessed in both surveys were of similar character but were phrased in slightly different ways. The goal of the questions was to address the same topics but represented from both a manager's and a technical operative's perspective.

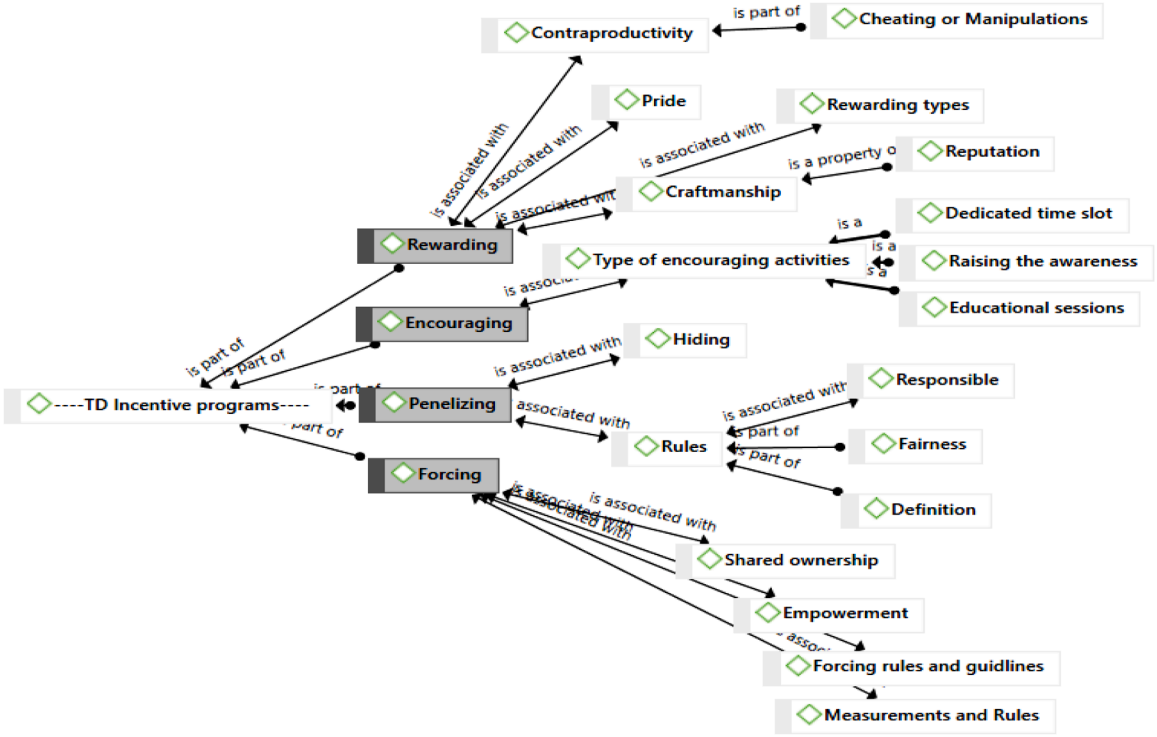

Fig. 3: Thematical Coding Scheme – Step 1.

The surveys included other questions on other topics related to TD management (designed with an additional purpose) such as processes, tools, etc., which are not relevant for this study (also including another survey design, e.g. using 4-Likert scales).

As illustrated in Table 2, we used the same statements but altered the phrasing depending on whether managers or technical roles were asked the questions.

3.2.2. Phase 8 – Analysis and synthesis (AS4)

This phase's data was collected using a similar approach as in phase 3, Step 1, where the survey data was analysed quantitatively. In this phase, the data from the managers and technical roles from the two sets of surveys were first analysed separately and then meta-analysed together.

3.2.3. Phase 9 – Qualitative data collection (DC4)

For this phase's data collection, four different semi-structured interviews were conducted using the same settings and approaches as described in Section 3.1.4.

We interviewed two interviewees with technical roles (developers), followed by three additional interviews with four managers (Project owner, Chief Product Owner, Process Manager, and Head of Architects). All interviewees had previously taken the survey in part 2 and the results from the surveys were presented to the interviewees during the interviews.

3.2.4. Phase 10 – Analysis and synthesis (AS5)

This stage focused on analysing the data collected in phase 9 using the same approach as in phase 5, Step1.

3.2.5. Phase 11 – Combined analysis and synthesis (AS6)

This phase combined the previous quantitative and qualitative data collection results using the same approach as in phase 6, Step 1. Tables 3–5

Table 2
Characteristics of the sample survey – all roles in Step 2.

Manager Roles phasing:	*Technical Roles phrasing:*	*Statements*
I encourage the software development teams to:	**My manager encourages my team to:**	• Avoid and Remove TD • Assess and report TD in the official backlogs to prioritise and remove it • Deliberate taking on TD if they get benefits from (e.g., to speed up delivery)
	My team colleagues encourage me to:	• Avoid and Remove TD • Assess and report TD in the official backlogs to prioritise and remove it • Deliberate taking on TD if they get benefits from it (e.g., to speed up delivery)
When is my team encouraged to remove TD?	**When is my team encouraged to remove TD?**	• Whenever they/we want • When they/we have extra time, budget, or human resources to be allocated • When they/we have a specific amount of time dedicated to TD removal (e.g., 10, 20%, etc.) • When they/we provide a business case for removing TD (e.g., reporting on costs, risks, and benefits of removing or keeping TD)

Table 3
Characteristics of the sample survey in Step 1.

Factor	Percentage split	Factor
Experience	< 2 years	3.90%
	2 - 5 year	10.50%
	5 - 10 year	17.40%
	> 10 years	68.20%
Roles	Developer/Program/Software Engineer	49.20%
	Software Architect	24.80% 6.20%
	Manager	6.20%
	Project Manager	5.0%
	Product Manager	5.0%
	Expert	3.50%
Team size	1–5 members	23.30%
	6–10 members	36.00%
	11–20 members	15.90%
	21–40 members	6.60%
	> 40 members	18.20%

Table 4
Examples of interview questions step 1.

TD Management Strategy	Examples of Interview Questions
Encouragement (RQ1)	• How do you perceive encouragement from managers for keeping the level of TD down? • How could such a strategy be implemented? • Do you agree with the results of the survey?
Rewarding incentive (RQ2)	• How do you perceive a rewarding incentive for keeping the level of TD down? • Do you have this or a similar strategy in place or planned for the future? • How could such a strategy be implemented?
Forcing (RQ3)	• How do you perceive a forcing mechanism for keeping the level of TD down? • Do you have this or a similar strategy in place or planned for the future?
Penalising disincentive (RQ4)	• How do you perceive a penalising disincentive for keeping the level of TD down? • How could such a strategy be implemented? • Does your company have a team and/or personal incentive/disincentive system for any other kind of quality criteria related to your software development process?
Other	• Which strategy of keeping the TD down do you consider to be the most/least successful (and why) and under what circumstances?

Table 5
Characteristics of the sample survey step 2.

Technical Roles			*Managers*	
	Percentage split	**Factor (%)**	**Percentage split**	**Factor (%)**
Experience	< 2 years	4.3%	< 2 years	11.5%
	2 - 5 year	6.5%	2 - 5 year	0%
	5 - 10 year	4.3%	5 - 10 year	7.7%
	10 – 20 years	84.4	10 – 20 years <	46.2%
	< 20 Years	0	20 Years	34.6%
Roles	Developer	13%	R&D manager	38.5%
	Team / FunctionalArchitect	8.7%	Product manager	26.9%
	Test / Quality Platform / Chief Architect	54.3%	CPO Product Owner Other managers	15.4% 11.5% 7.7%
Team size	**Size of team:**		**Managing numbers of teams:**	
	1–5 members	23.9%	1–2 teams	53.8%
	6–10 members	23.9%	3-5 teams	11.5%
	11–20 members	52.2%	6–10 teams	7.7%
	> 20 members	0	11-15 teams	19.2%
			>15	

4. Results and findings

This section presents the results of this study where the first four subsections (4.2 - 4.5) present the result for RQ1-4, followed by Sections 4.6 and 4.7, which address RQ 5-7. For more in-depth details about the result in this step of the study, we refer to our earlier publication [41].

In the survey used in Step 1, the participants were asked to rate their agreement with four statements

(ST1-ST4) using a 6-point Likert Scale. The ratings provided by the respondents for each of the survey statements are presented in Fig. 4 and further described in the following Sections 4.2 to 4.5.

4.1. Demographics data

In the first step of the study and across all collaborators, 258 respondents answered all questions. The demographics detail data is reported in Appendix A. Characteristics of the sample survey in this step are presented in Table 5.

4.2. Encouraging strategy (RQ1)

The first research question investigates how common an *encouraging strategy* is to keep the level of TD down and how software engineering practitioners perceive this TD management strategy.

4.2.1. 4.2. Survey results

When looking at the results from the different statements in Fig. 4, it is evident that one of the statements excels compared to the others.

What stands out in the Fig. is the statement assessing whether the respondents are encouraged to keep the level of TD down (ST4) where 151 respondents i.e. 60.3% (9.3% strongly agree, 24.4% agree, 26.6% somewhat agee) of the respondents agree to some extent that they are encouraged, and 97 respondents i.e. 39.7% disagree to some extent that they are encouraged to keep the level of TD down.

A remarkable result of this statement is that 29 respondents, i.e. 11.9%, strongly disagree that they are encouraged to keep the level of TD down.

4.2.2. Effective or desirable strategies (RQ1.1)

The practitioners' attitudes towards introducing TD or conducting TD remediation tasks were described as being guided and targeted by the mindset and the attitudes of the management where recognition of leaders and peers was important. This meant that when management focused its attention on the importance of TD, the employee was encouraged to focus his or her work in the same direction.

All the interviewees considered an encouraging managing strategy concerning TD to be highly effective and impactful. Several of them described that this strategy could clearly have more emphasis within their organisations and thereby, also have a more significant impact on the amount of TD in their software.

4.2.3. Tactics for encouragement

Several of the interviewed companies strived to continuously raise awareness about the concept of TD and its related negative consequences as an encouragement to keep the level of TD down. Some companies ran satisfyingly dedicated educational sessions to explicitly address how to avoid the introduction of TD in the first place.

The managers of some teams in one of the interviewed companies had set aside a specific amount of working time within each sprint to allow for explicitly spending time on TD remediation activities (without imposing this on the developers) together with other software-improving activities. This dedicated time slot encouraged the involved engineers (such as testers, developers, and architects) to focus on TD issues in every sprint as an incorporated part of their overall working process.

4.3. Rewarding incentives (RQ2)

The second research question addresses how common rewarding incentives are to keep the level of TD down and how software engineering practitioners perceive this TD management strategy.

4.3.1. Survey results

As illustrated in the first statement (ST1) in Fig. 4, only 35 respondents, i.e.14.3%, of the respondents agree to some extent (0.8% strongly agree, 3.7% agree, 9.8% somewhat agree) with being explicitly rewarded when keeping the level of TD down. Thus, 212 respondents, i.e. 87.7% of the respondents state that they are not explicitly rewarded for this. What stands out in this data is that only two (2) respondents state that they strongly agree with being rewarded when keeping the level of TD down; meanwhile, a hundred (100) respondents state that they strongly disagree with being explicitly rewarded if they keep the

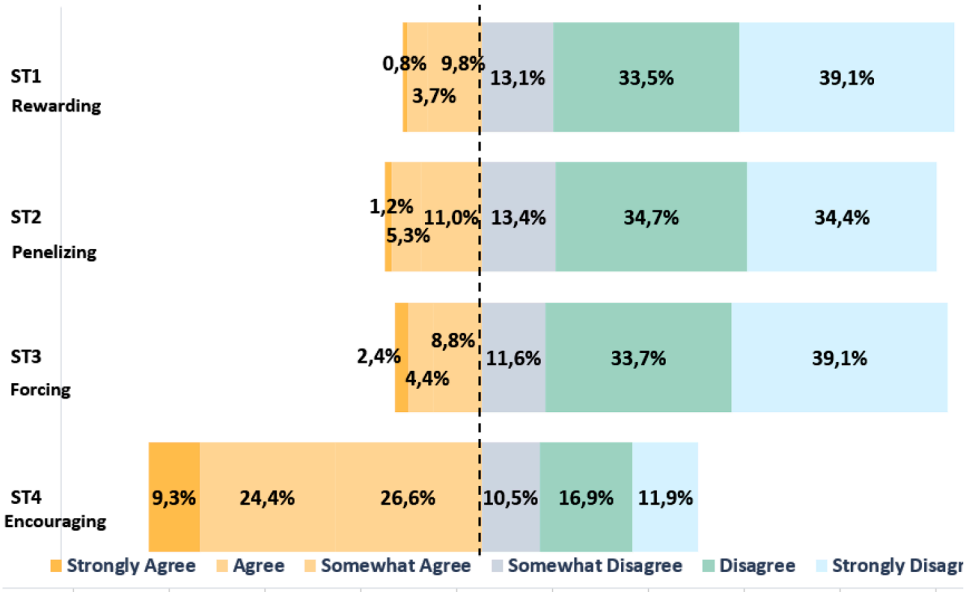

Fig. 4. Summary of the responses to the survey statements in Step 1.

level of TD down.

4.3.2. Effective or desirable strategies (RQ2.1)

None of the interviewed companies had an incentive programme where employees were rewarded for any explicit behaviour (not just specifically TD). The interviewees' thoughts differed as to whether adopting rewarding incentives is an effective or desirable strategy to keep the level of TD down.

Some interviewees were skeptical about explicitly highlighting and rewarding specific working activities and behaviours such as TD remediations since they thought keeping the level of TD down should be an activity that comes with the craftsmanship of software development and the working pride of software engineers. On the other hand, several other interviewees argued that a TD managing strategy based on rewards could be effective since rewards can motivate practitioners to manage TD further.

4.3.3. Tactics for rewards

Concerns regarding the different appropriate types of rewards were widespread. Some proposed rewards such as monetary compensation, extra holidays, and pizzas to the teams (since none of them had any incentive programs in place). Meanwhile, other interviewees said a reward does not have to be tangible; it could be a simple acknowledgment since official praise and, therefore, an enhanced reputation is considered equally important: "*A reward does not have to be money. You could achieve some level of reputation that will actually be enough of a reward in itself*."

Nevertheless, even if a rewarding incentive has the best intention of decreasing the amount of TD in the software, such initiatives could easily be misused by causing a counterproductive backlash where, for example, practitioners primarily focus on TD remediation tasks to get rewards or only focus on the TD items that are easy to refactor. Therefore, they would focus less on other tasks and goals, which can harm the overall implementation or delivery of the software.

Yet another concern that was expressed by several interviewees refers to the possibility of manipulating such reward systems by first introducing a large amount of TD and then refactoring this to get the reward.

Taken together, if an incentive programme for TD remediation should be introduced, such a programme must be carefully designed to avoid counterproductive results that instead generate even more TD; also, it is important to incorporate an impartial way into the design that is not easy to manipulate.

4.4. Forcing mechanisms (RQ3)

The third research question aims to assess the forcing mechanism to keep the level of TD down and how software engineering practitioners perceive this TD management strategy.

4.4.1. Survey results

When assessing whether the respondents are being forced to keep the level of TD down, the result from the second statement (ST3) in Fig. 4 shows that 39 respondents, i.e. 15.6%, agree to some extent with this statement (2.4% strongly agree, 4.4% plus agree, plus 8.8% somewhat agee) and that 84.4% of the respondents disagree to some extent with being forced to keep the level of TD down.

4.4.2. Effective or desirable strategies (RQ3.1)

None of the interviewed companies had any forcing rules or requirements related to TD. Notably, all the companies had other forcing rules related to their software development processes such as following code standards, documentation requirements, and performing tests. These rules applied primarily to specified mandatory activities and requirements that had to be fulfilled for the delivery to be viewed as complete and further activities to take place.

Even if the TD and its negative effects are known to the software engineers, it can be challenging to get the time and budget from management to refactor the software. Here, a positive side to forcing a TD management strategy was described in terms of empowerment. Such a strategy would give the practitioners authority to conduct mandatory TD remediation tasks without arguing and motivating managers to perform the action. Yet another finding was that forcing TD remediation activities seems to become more vital for companies adopting shared ownership of their software product portfolio where several teams collaborate on the same software without having strict ownership of the components.

4.4.3. Tactics for implementing a forcing strategy

Another view on the enforcement of TD activities was described as a transition from an encouraging strategy to a forcing strategy.

Several interviewees recommended that a company should first focus on encouraging initiatives and if such a strategy were conceived as not enough, this forcing strategy could be implemented. Directly implementing a forcing strategy was not recommended by the interviewed companies. One company described its history as moving from an encouraging strategy to adopting a forcing strategy (however, the target here was related to TD): *"It depends on the scale on the organisation. A couple of years ago, we didn't force that much. We were encouraged, and that was because we had some sort of concept of ownership. I would say that pride in our product was sort of our encouragement. It was your baby and you wanted to be proud of it. This is what partially drives you forwards."*

4.5. Penalising disincentives (RQ4)

The fourth research question set out to investigate how common *penalising* disincentives are to keep the level of TD down and how software engineering practitioners perceive this TD management strategy.

4.5.1. Survey results

Looking at the third statement (ST2) in Fig. 4, it is apparent that 43 respondents, i.e 17.5%, agree to some extent (1.2% strongly agree, 5.3% agree, 11% somewhat agree) with being explicitly penalised when not keeping the level of TD down. Therefore, 205 respondents, i.e., 82.5%, state that they are not penalised for failing to do so.

4.5.2. Effective or desirable strategies (RQ4.1)

Even if the survey showed that respondents are being penalised, none of the interviewees in the study were familiar with any penalising activities within their companies, using, for example, monetary fines and salary reductions. All of the interviewees had direct negative attitudes towards implementing a TD management strategy based on penalising practitioners or teams who fail in keeping the level of TD down.

4.5.3. Tactics for penalisation

There are several different issues to consider if a penalising strategy should be implemented to facilitate the management of TD. Several interviewees raised the importance of establishing fair, adequate, and succinct rules that should be clearly conveyed, understood, and followed by all employees concerned.

4.6. The perception of encouragement - survey results

This section reports the findings from the **second step** of the study, based on the first step's findings. One of the key findings in the first step was that about 60% of the survey respondents stated that they were encouraged by managers to keep the level of TD down (see Section 4.2.1). This section addresses the research questions RQ5, RQ6, and RQ7, which all provide a more detailed assessment and an additional in-depth analysis of the TD management strategy based on encouragement.

Since encouragement is a human activity based on communication that involves both a "sender" and a "receiver", wherein the sender

conveys a message, and the receiver gets that message, we have asked both managers and technical roles to state to what extent they perceive that they send the message (managers) and to what extent they receive the message (technical roles).

4.6.1. The perception of receiving or providing encouragement (RQ5)

This sub-section addresses the fifth research question to understand the extent to which different specific TD management activities are encouraged and *who* encourages these activities, using three different role perspectives:

a) How managers perceive providing encouragement to technical roles
b) How technical roles perceive receiving encouraged by managers
c) How colleagues with technical roles encourage each other within the teams to address TD.

Furthermore, to provide a higher granularity for the encouragement of TD management activities, we collected data using the three different TD encouraging activities:

1) Avoid and remove TD
2) Assess and report TD in the official backlogs to prioritise and remove it
3) Deliberately take on TD if they get benefits from it (e.g., to speed up delivery)

The respondents' quantitative summary statistics for each of the survey statements are presented in Fig. 5 and further reported and described together with the qualitative results from the interviews in the following sections.

Encouraged to avoid and to remove TD: The first three upper stack bars in Fig. 5 (ST18, ST8, and ST5) address the encouragement to *avoid or remove TD*. By looking at these bars, it is apparent that the results illustrating this activity differ significantly between the different groups of respondents.

As illustrated in the first stack bar (ST18), about 91% (19.1 % strongly agree, 71.4 % agree) of the managers who took the survey agree that they should encourage software development teams to avoid or remove TD. In comparison to this, "only" 46% of the technical roles (in bar ST8) (2.3% strongly agree 44.2% agree) perceive that they are encouraged by their managers. However, about 79% of the technical roles (in bar ST5) (16.3% strongly agree, 62.8% agree) perceive that they are encouraged by their team colleagues to avoid or to remove TD.

Our chi-square test of independence, yielding a very low p-value (0.0003605), which statistically confirms that there is a significant difference between the answers given by managers and technical roles.

This is a rather remarkable result indicating that the managers perceive that they encourage the teams with a technical role to avoid or remove TD. However, the technical roles do not seem to receive this encouragement to the same extent. However, it is apparent from the results that the technical roles perceive they are encouraged by their team colleagues to avoid and remove TD.

Assess and report TD in the official backlogs to prioritise and remove it: As illustrated in Fig. 5, bar ST19, about 86% of managers (23.8% strongly agree, 61.9% agree) stated in the survey that they encourage the teams to assess and report TD in the official backlogs to prioritise and remove it.

When surveying the technical roles to see to what extent they perceive receiving encouragement from managers to perform this activity, about 56% of the technical roles (in bar ST9) (4.7% strongly agree, 51.2% agree) perceive that they are encouraged by their managers, while also 14% of the respondents strongly disagree with this statement.

Furthermore, by looking at the results in bar ST6, it is apparent that about 72% (18.6% strongly agree 53.5% agree) of the technical roles perceive that they are encouraged by their team colleagues to avoid or to remove TD and to assess and report TD in the official backlogs.

Our chi-square test of independence, yielding a very low p-value (0.0007684), which statistically confirms that there is a significant difference between the answers given by managers and technical roles.

Deliberate taking on TD if they get benefits from it: As shown in Fig. 5, bar ST20, about 67% (4.8% strongly agree, 61.9% agree) of managers stated in the survey that they encourage their teams to deliberate taking on TD if they get benefits out of it. This encouragement from the managers was received by 56% of the technical roles (in bar ST10), and 65% of the technical roles also perceive that they are encouraged in this activity by their team colleagues (bar ST7).

Despite these visual observations, our chi-square test of independence did not find a statistically significant difference between the answers of different roles, although the p-value (0.1418) is rather low.

When summarising and analysing the three different activities together, it becomes clear that managers perceive they encourage the technical roles to *avoid and remove TD* to the greatest extent. However, this result is thought to be quite contrary to how technical roles perceive this activity since it is actually perceived by those in technical roles as being the least encouraged by managers among the three different listed

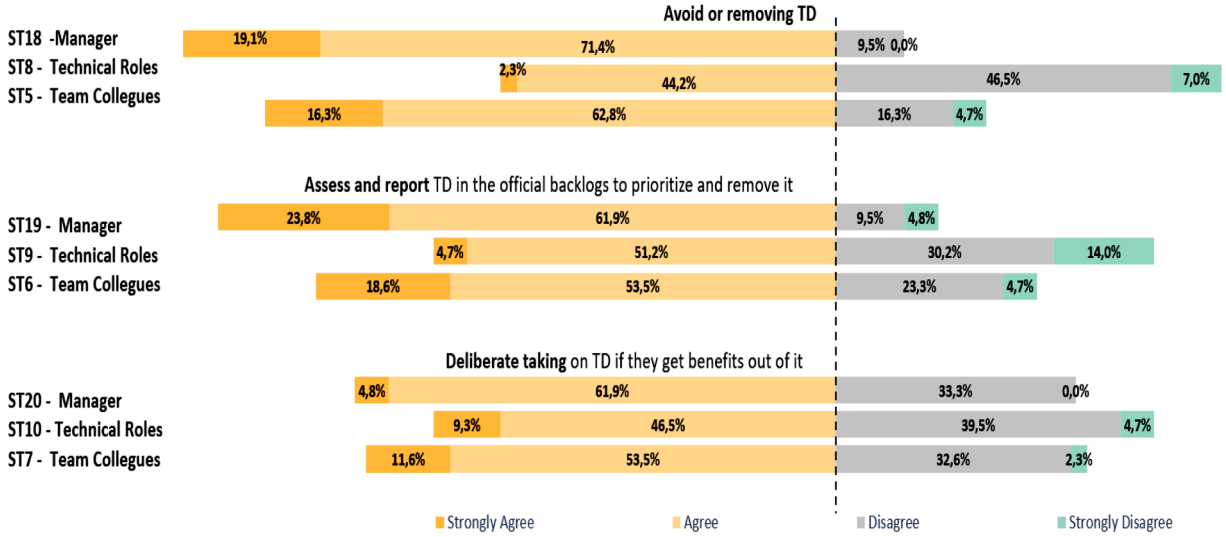

Fig. 5. Summary of the responses to the survey statements reflecting the perception of receiving or providing encouragement.

activities. However, this activity seems to be the most common activity that team colleagues encourage each other to address.

Furthermore, the activity that managers most strongly agree to was that they encourage the technical roles to assess and report TD in the official backlogs (23.8% strongly agree). Meanwhile, as seen from the technical role perspective, this activity was the one they most strongly disagree with in terms of being encouraged to address it (14% strongly disagree).

Taken together, these results suggest that there is quite an extensive and misalignment between how managers perceive that they encourage the technical roles to address TD and how the technical roles perceive being encouraged by their managers. However, it is clear that the technical roles encourage each other to address TD to a relatively large extent. It is notable that both the interviewed developers and the interviewed managers came from the same departments and worked in and shared the same working environment and settings.

Individual professions – grouped as technical roles (RQ5)

To further study how the technical roles' teams are encouraged by their managers to address TD, this section presents the survey results from each of the individual roles in the earlier reported results where they were analysed as a group. These results are illustrated in Fig. 6.

The distinct roles that we earlier grouped as "technical roles" (in Section 4.6.1) are a) developers, b) team-functional architects, c) test and quality engineers, and d) platform-chief architects.

Further, we use the same TD encouragement activity as earlier reported in Section 4.6.1.

Encouraged to avoid and remove TD: The first four upper stack bars in Fig. 6 (ST8) visualise the extent to which the different individual roles agree with their teams being encouraged by their managers to avoid or remove TD.

From the data, it is apparent that the different investigated professions perceive being encouraged (from managers) quite differently to avoid or remove TD. What stands out when comparing the result is, for example, that none (0%) of the developers strongly agree with this statement and that only 20% of them agree to be encouraged by their managers to avoid or to remove TD.

Furthermore, when looking at these results, it is also apparent that test and quality engineers perceive being encouraged to a limited extent by their managers, where 25% agree and the remaining 75% disagree with being encouraged to avoid or removed TD.

Meanwhile, about 64% of the team and functional architects agree (9.1% strongly agree plus 54.5% agree) to the same statement. However, this profession's results are quite spread out, where, for example, 18.2% strongly disagree with this statement.

Assess and report TD in the official backlogs to prioritise and remove it: As illustrated in Fig. 6, bars ST9, 0% of both the developers and the team and functional architects strongly agree that they are encouraged to assess and report TD in the official backlogs and also quite a few of these professions strongly disagree with this statement (40% as opposed to 27.3%). However, three-thirds (75%) of the surveyed test and quality engineers agree to be encouraged by their managers to assess and report TD in the official backlogs.

Deliberately taking on TD if they get benefits out of it: As shown in Fig. 6, bars ST10, 60% of the developers perceive that they are being encouraged by their managers to deliberate taking on TD if they get benefits from it (0% strongly agree plus 60% agree). However, the results of this activity show that only about 27% of the team and functional architects agree (9.1% strongly agree, 18.2% agree) with this statement. Also, 18.2% of them strongly disagree with being encouraged to deliberately take on TD if they benefit from it.

Taken together, when looking at these results, it is clear that different individual roles perceive the extent of their managers' encouragement differently, where, for instance, developers commonly strongly disagree with all three statements. However, a word of caution is needed here since, despite these observations, our chi-square test of independence did not find a statistically significant difference between the answers of different roles.

4.6.2. *The situation when the teams are encouraged to remove TD (RQ6)*

This section reports the survey results, addressing different situations or under what circumstances when the managers perceive that they encourage the teams to remove TD and receive such encouragement.

As illustrated in Fig. 6, we assess four different situations and circumstances where the encouragement of removing TD may take place; a) whenever the team wants, b) when the team has extra time, budget, or human resources, c) when the team has a specific amount of time dedicated to TD removal, and d) when the team provides a business case for removing TD.

Encouraged to remove TD whenever we/the team want: As shown in Fig. 7, bar ST24, about one-third of the managers perceive they encourage the teams to remove TD (4.8% strongly agree, 28.6% agree) whenever they want. This view is shared by about 21% of the technical roles (4.76% strongly agree plus 16.67% agree), as illustrated in ST11. However, one should also note that a substantial part of the technical roles (about 79%) disagree with being encouraged by managers to remove TD whenever they want (47.6% disagree, 31% strongly

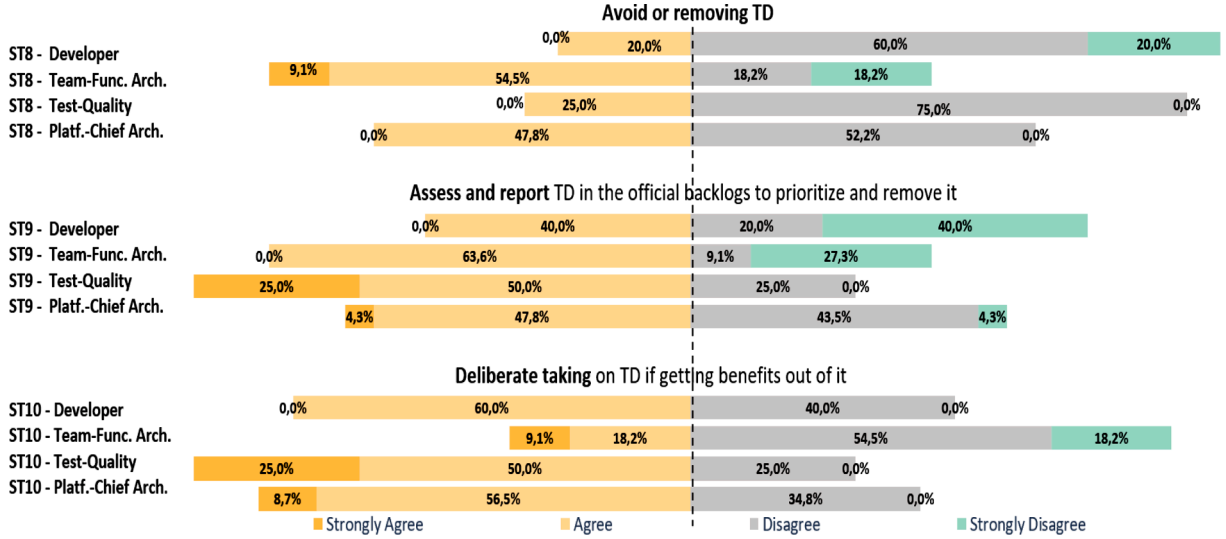

Fig. 6. Summary of the individual professions responses to the survey statements reflecting the perception of receiving encouragement.

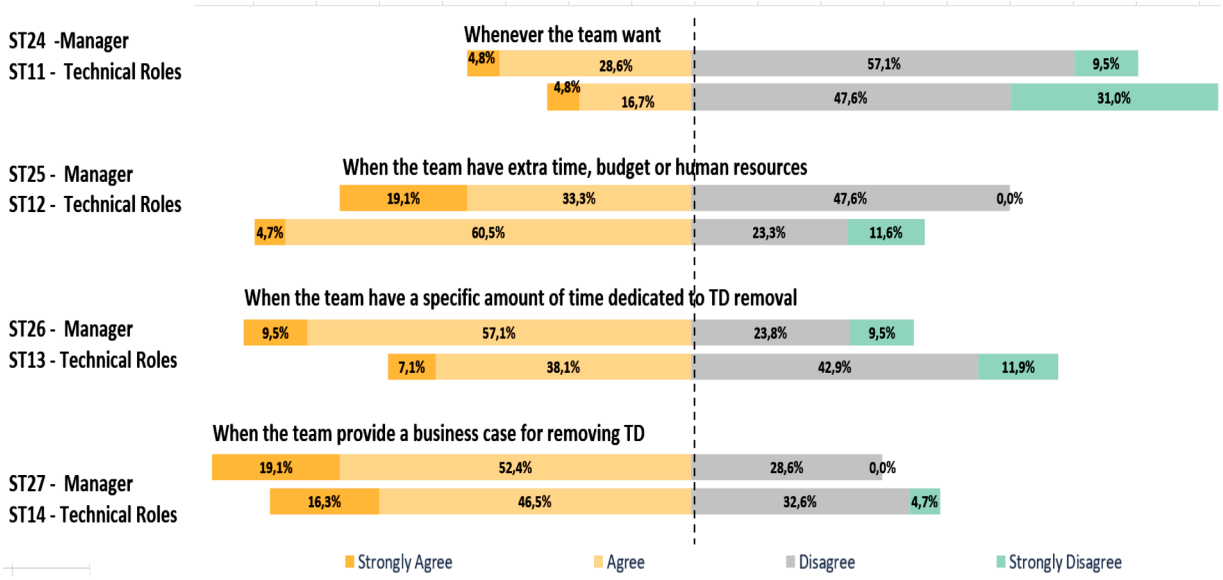

Fig. 7. Summary of the responses to the survey statements reflecting the situation when teams are encouraged to remove TD.

disagree).

Encouraged to remove TD when we/the team have extra time, budget, or human resources: When assessing the result of ST25 and ST12 in Fig. 7, the results show quite interestingly that the technical roles perceive receiving more encouragement to remove TD when they have extra time, budget, or human resources compared to what the managers report that they actually encourage the teams to. This result is the only one among all the assessed statements where the managers perceive that they encourage less than how the technical roles perceive their managers encourage them.

Encouraged to remove TD when we/the team have a specific amount of time dedicated to TD removal: Bar ST26 and ST13 in Fig. 7 shows that about two-thirds of the managers agree that they encouraged the teams to remove TD when they have a specific amount of time dedicated to TD removal (9.5% strongly agree plus 57.1% agree). However, only about 45% of the technical roles report that their managers encouraged them to do so (7.1% strongly agree plus 38.1% agree), meaning that about 55% of the technical roles disagree with this statement.

Encouraged to remove TD when we/the team provide a business case for removing TD: The last two stack bars (ST27 and ST14) in Fig. 7 report the result related to the encouragement of removing TD when the teams provide a business case for doing so. By looking at these stack bars, it is evident that by comparing this result with the previous comparisons, the perception of these statements seems to differ less between the managers and technical roles. ST27 shows, for instance, that 71% of the managers agree (19.1% strongly agree, 52.4% agree) that this action should be encouraged. Meanwhile, about 63% of the technical roles perceive this type of encouragement from their managers.

Our chi-square test of independence did not find statistically significant difference between the answers of managers and technical roles, although for ST24 and ST11 (related to *Encouraged to remove TD whenever we/the team want*), the p-value is 0.06097, which is very close to the 0.05 (usually used α).

When analysing the four different activities together, the results indicate that the technical roles perceive that they are more encouraged to remove TD by their managers when they have extra time or resources available or when they can provide a business case for TD refactoring activities. Moreover, the technical roles seem to perceive less encouragement from their managers to remove TD whenever they want.

Taken together, when analysing the statements which relate to when the TD refactoring is encouraged, it is evident by the presented findings that there is a misalignment between the perception between the manager perspective compared to the technical role perspective, related to the different situations and circumstances when conducting TD refactoring activities are encouraged.

4.7. The perception of encouragement: qualitative results and discussion

After the quantitative data was collected and analysed, the results were presented to a subset of the respondents during four different interviews with both technical roles and managers (separately).

The survey results were described as quite surprising by two of the interviewed managers, where, for example, one Chief Product Owner (CPO) said, "*I would not expect that they [the managers] put so much effort into encouraging TD management.* Another interviewed manager offered a potential explanation for the relatively high percentage of the perception of encouragement from the management as "*The managers themselves think they encourage TD management a lot since it is a very high percentage…I think the managers 'want' to have this attitude of encouraging the removal of TD.*"

Moreover, during these follow-up sessions, one of the interviewed managers concluded the overall results as "The perception of the managers and the technical people is completely different, but the teams show some awareness, so they [the teams] deal with it [TD] anyway."

Contrary, during the interviews with respondents having technical roles, concerns were expressed about the encouragement of addressing TD from their managers' side. Even if they perceived that they received some encouragement from their managers, the TD-related tasks were commonly down-prioritised in favour of implementing new features instead, which was perceived by technical roles (especially developers) as a lack of encouragement.

When interviewing Chief Product Managers (CPMs), it was revealed that they often do not agree to prioritise large TD issues (for example, related to architecture refactoring) because of the lack of a good business case to support it, which should be provided by technical roles (especially architects) and because of the lack of a suitable solution proposal to remove the TD. This was reported by a CPM with a technical background, which decreases the likelihood that, as proposed by other technical roles in the interviews, managers would down-prioritize TD refactoring because of a lack of technical understanding.

When asking what kind of TD was encouraged to be removed, we found a difference that could explain the divergence of perception between managers and technical roles. Both managers and technical roles

revealed that they encouraged and are encouraged respectively to remove TD that hinders the implementation of the features in the immediate future. In most cases, this is represented by “small” TD issues, or other issues that can be fixed by dedicating a limited percentage of development time (e.g., 10-20%). However, such time is not enough to remove larger TD, such as the architectural type. Large architectural refactoring, where packages of epic dimension are often required and should be prioritised against features at a high management level is still often down-prioritised. Consequently, managers feel that they encourage staff to remove (small) TD, while technical roles feel that they are not encouraged to remove (large) TD. This difference in perceiving encouragement might have caused the misalignment reported in the data.

Another issue brought up by POs during the interviews was that TD needs to be prioritised over interests from several stakeholders in complex projects. The projects’ complexity often leads to wrongly estimated time to implement the features: then, the extra time needed for the features ends up, in practice, decreasing any time reserved to refactor TD. This means that even if a percentage of time is reserved to remove TD, it is not actually enough and the pressure to implement features is too high. This infers that TD should be prioritized at a higher management level and when the projects’ budget is decided upon. Our results further reveal that TD is not, in fact, often taken into consideration as a variable when project resources are allocated.

Taken together, the misalignment of encouragement between the managers and the technical roles may be because managers, in general, perceive that they encourage the technical roles and even if the technical roles receive this encouragement, they are not provided with additional time and resources to actually address TD in reality.

However, with limited interviewees, caution must be applied as the findings might not be transferable to all different technical roles and all manager roles.

5. Discussion and limitations

One of the main implications for TD research is a need for more and better empirical studies that may assist in finding strategies to keep the level of TD down. Since no research to date was found in the SE research field on how different TD management strategies can be applied to keep TD down, this research provides novel results that may contribute to this.

In the first step of the study, four main strategies were identified in the initial literature research, which was particularly interesting when portraying how software managers can influence and impact the software engineers’ attitudes and working behaviour with TD.

As illustrated in Fig. 8, we propose a model describing the different identified and investigated TD management strategies. The model spans four quadrants and is named "The Four TD Management Quadrants", which outlines that TD managing strategies can be either of an incentive/disincentive nature, focusing on either a desired or undesired behaviour.

This model considerably expands our understanding of different TD management strategies, together with their different modalities and tactics, and it also describes both how the strategies relate to each other and how they differ. This model can assist managers in deciding which strategy to adopt and support transition plans, shifting from one strategy to another. It also illustrates different TD management strategy implications for the teams.

Among the different studied strategies, the result shows that today’s software companies most commonly use a TD management strategy based on the encouragement of employees, where 60% of the respondents in the survey state that they are, to some extent, encouraged to keep the level of TD down. Meanwhile, the other investigated strategies such as using a strategy based on forcing mechanisms or adopting incentive or disincentive programmes were rarely used by the companies.

Furthermore, among the investigated companies, there was a strong belief that both the encouraging and the reward TD management strategy would be valuable to further decrease the amount of TD in the software; meanwhile, the forcing and penalising strategies were not considered as desired and constructive.

One motivating finding is that practitioners conceive that the attitudes and mindset toward TD remediation tasks from their managers significantly impact the way they address TD. Further, and perhaps the most striking results in this study are that a TD managing strategy based on encouragement has a significant impact on the way practitioners work with TD.

However, since the result shows that quite a lot of the respondents (40%) to some extent still do not agree with being encouraged to focus their effort on TD remediation tasks, this result shows that there is an unfulfilled potential for managers to impact how practitioners can reduce TD by adopting a TD management strategy based on

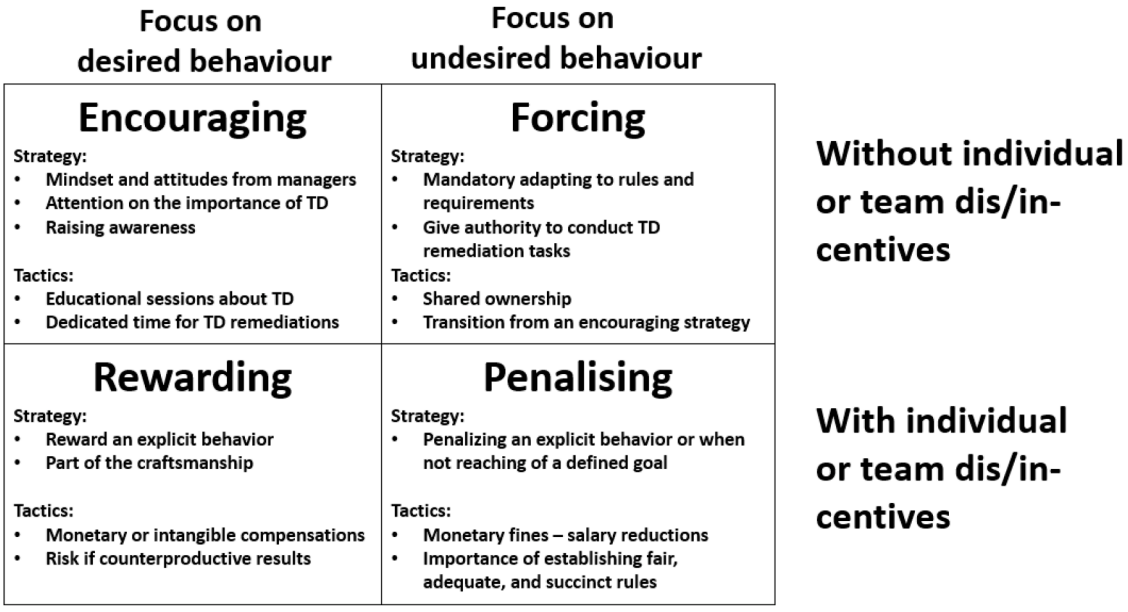

Fig. 8. The Four TD Management Quadrants Model.

encouragement and without having to introduce forcing mechanisms or strategies based on rewards or penalisations.

The second step of the study primarily focused on the output from the first step to provide more in-depth information about *how, when, and by whom* TD management is encouraged.

It is evident from this step of the study that there are significant differences in how managers perceive that they encourage the technical roles compared to how the technical roles perceive being encouraged by the managers. Commonly, the managers perceive that they encourage addressing TD significantly more often compared to how the technical roles perceive receiving this encouragement from their managers. The result also indicates a misalignment in when or under which circumstances TD refactoring activities should be carried out.

Several different factors could explain the misalignment of TD management's encouragement between the technical roles and their managers. In particular, we found that the higher-level managers indeed encourage teams to remove short-term and smaller TD issues. However, they do not often prioritise large TD issues to be refactored. This happens because of a lack of business cases and viable refactoring solutions. In a sense, this can also be seen as the *managers not feeling encouraged enough to prioritise TD refactoring* at a higher level. In turn, the down-prioritisation causes the technical roles (especially developers) to also feel not encouraged to remove TD due to lack of dedicated time.

These relationships are illustrated in Fig. 9, where the current successful (green arrows) and unsuccessful (red arrows) encouraging practices across the different roles are illustrated.

Although architects and POs agree that TD is an important area for regulation, they report struggling to provide the right business motivation for TD to be prioritised (for example, quantifying the interest in TD). In addition, feature pressure and the complexity of the projects and stakeholders tend to cause unplanned work that then also eats up the time dedicated to remove TD.

Given the reported results, we foresee four possible encouragement practices, which may act as implications for both industry and academia, even if they would benefit from further investigation:

1) Technical Debt should be recognised as a variable in project planning in order to better protect time for the eventual and unavoidable occurrence of TD.
2) Architects and POs need to motivate higher-level management to prioritise the removal of large TD items. To do so, more emphasis can be placed on creating business cases and quantifying the TD interest.
3) High-level managers should prioritise larger TD issues against the features to convey the message to the developers that TD is indeed an important entity to be taken into consideration.
4) We also found that companies are often unaware of the extent of misalignment between employees' perceptions and top management. Once this gap was showed to the participants, they were keener to revise their encouragement strategy.

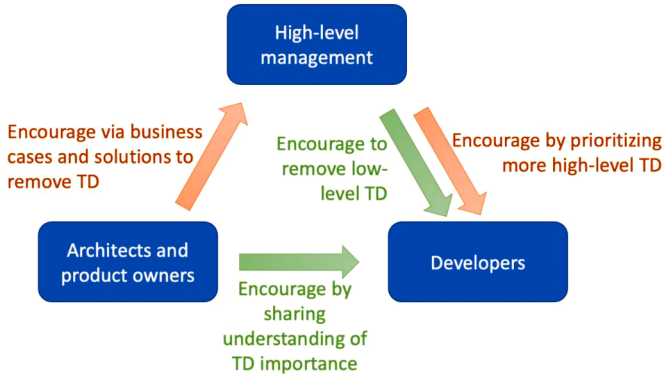

Fig. 9. Encouragement strategies between roles.

In conclusion, the success of encouragement across the roles seems to be closely related to how the TD is communicated and prioritised.

6. Threats to validity

Several vital threats to validity necessitate a cautious interpretation of the results of this study. We have chosen a classification scheme to distinguish between different aspects of validity and threats to validity provided by Runesson and Höst [35]. This scheme includes four aspects of validity: construct validity, internal validity, external validity, and reliability.

Construct validity reflects the extent to which the studied operational measures represent what the researchers have in mind and what is investigated according to the stated research questions [42]. This is commonly one of the main threats to validity in surveys, as the respondents might interpret the survey questions and other terms differently. To mitigate this threat, we provided the respondents with the following description of TD before answering the questions: "*Technical Debt is a metaphor that describes a real-life phenomenon and it provides a way of talking and reasoning about difficulties related to software development and software maintenance. Below is a brief description of what Technical Debt is: Technical Debt (TD) is usually described as the non-optimal code or other artifacts related to software development that gives a short-term benefit but causes a long-term extra cost during the software life-cycle.*"

Moreover, and as described in Section 3, this study may also potentially and unwittingly have left out management strategies that were not found during the construction of the conceptual framework.

Furthermore, this study could possibly suffer from internal validity by affecting our ability to explain the phenomena that we accurately observed [21]. However, to mitigate this threat, we triangulated both surveys' findings by conducting follow-up interviews validating the derived results.

Additionally, to minimise the threat of misunderstanding the different topics in the survey, we initially conducted two pilot studies (one for each survey) with industrial practitioners. Understanding the terms was also addressed during the follow-up interviews. External validity focuses on the extent to which it is possible to generalise the findings. There is always a risk in surveys that the sample is biased and for this topic, a potential threat refers to the demographic and cultural distribution of response samples. As reported in Section 3.1.2, we mainly investigated companies from the Scandinavian area in step 1, which may have a cultural impact on respondents' experiences and views on penalising, rewarding, forcing, and encouraging management actions. The results could, therefore, potentially be different in other cultural or geographical areas. Therefore, further work is needed to replicate the results in other geographical areas and other software development cultures. However, to mitigate this validity issue, we attempted to enlarge the respondents' sample by inviting additional participants globally via LinkedIn. Reliability addresses whether a study would yield the same results if other researchers replicated it. In this sense, triangulation is important [42], and to mitigate this threat, we have employed source triangulation (several companies and several professional roles), methodological triangulation (quantitative analysis based on surveys and qualitative analysis based on interviews), and observer triangulation (all authors participated in the analysis).

7. Conclusions

This study investigates how common different management strategies are when managing TD and investigating how software practitioners perceive such TD management strategies. More specifically, we study how software management influences how software practitioners work with TD, for example, by continuously encouraging and rewarding those who focus on TD remediation and limitation activities. Yet another TD management strategy we examine in this study is based on penalisation and forcing mechanisms. The results show that software

practitioners are not commonly rewarded, penalised, or forced to keep the level of TD down.

Furthermore, the result shows that a TD management strategy based on encouraging activities is described as having a significant impact on the attitudes and behaviours of software engineers when addressing TD.

The results from both the first and the second step of this study show that an extensive number of the respondents state that they are not directly encouraged by managers to keep TD down. This indicates that there is considerable unfulfilled potential to influence how software practitioners can limit and reduce TD by adopting a TD management strategy based on encouraging activities where, for example, the concept of TD is acknowledged and recognised more broadly.

Moreover, this study also contributes to a TD management quadrant model describing four different TD management strategies and its tactics together with recommendations on how to implement such strategies in practice.

Taken together, besides indicating the importance of managers encouraging the technical roles to address TD, it is also important that managers dedicate extra time and resources so that the technical roles actually may conduct these activities in reality.

Finally, this conclusion presents many opportunities for future work. A singular study is insufficient to build a solid theory covering all different strategies, thus we encourage others to replicate our study under similar or different settings to also include other strategies besides the encouraging strategy.

CRediT authorship contribution statement

Terese Besker: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization, Project administration. **Antonio Martini:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing – original draft, Visualization. **Jan Bosch:** Conceptualization, Methodology, Supervision.

Declaration of Competing Interest

The authors (Besker, Martini and Bosch) declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A

Survey Questions in Step 1 (used in phase DC1)

How many years of experience in the Software Development area do you have?

- < 2 years
- 2 - 5 year
- 5 - 10 year
- 10 years

What is your job role?

- Developer/Program/Software Engineer
- Software Architect
- Manager
- Project Manager
- Product Manager
- Expert

How big is your software development team?

- 1–5 members
- 6–10 members
- 11–20 members
- 21–40 members
- > 40 members

Please indicate your level of agreement or disagreement with each of these statements regarding your organizational strategy to manage Technical Debt.

- Using Likert scale: Strongly agree, Agree, Somewhat agree, Somewhat Disagree, Disagree, Strongly Disagree
- Our team is or I am explicitly rewarded if TD is kept down.
- Our team is or I am explicitly penalised if TD is not kept down.
- Our team is or I am explicitly forced to keep the level of TD down (i.e., to be allowed for deployment)
- Our team is or I am explicitly encouraged if TD is kept down.

Survey Questions in Step 2 (used in phase DC3)

What is your experience with software development?

- < 2 years
- 2 - 5 year
- 5 - 10 year
- 10 - 20 years
- < 20 Years

What is your role?

Ø Technical roles:
  - ○ Developer
  - ○ Team/FunctionalArchitect
  - ○ Test/Quality
  - ○ Platform/Chief Architect
  - ○ Managers roles:
  - ○ R&D manager
  - ○ Product manager
  - ○ CPO Product owner
  - ○ Other managers

What is the size of your team?

- 1–5 members
- 6–10 members
- 11–20 members
- > 20 members

Asked in the survey version to managers:

Using the Likert scale: Strongly agree, Agree, Disagree, Strongly disagree

- **I encourage the software development teams to:**
  - Avoid and Remove TD
  - Assess and report TD in the official backlogs to prioritise and remove it
  - Deliberate taking on TD if they get benefits from it (e.g., to speed up delivery)
- **When is my team encouraged to remove TD?**
  - Whenever they/we want
  - When they/we have extra time, budget, or human resources to be allocated
  - When they/we have a specific amount of time dedicated to TD removal (e.g., 10 or 20%, etc.)
  - When they/we provide a business case for removing TD (e.g., reporting on costs, risks, and benefits of removing or keeping TD)

Asked in the survey version to technical roles:

Using the Likert scale: Strongly agree, Agree, Disagree, Strongly disagree

- **My manager encourages my team to:**
  - Avoid and Remove TD
  - Assess and report TD in the official backlogs to prioritise and remove it
  - Deliberate taking on TD if they get benefits from it (e.g., to speed up delivery)
- **My team colleagues encourage me to:**
  - Avoid and Remove TD
  - Assess and report TD in the official backlogs to prioritise and remove it
  - Deliberate taking on TD if they get benefits out of it (e.g., to speed up delivery)
- **When is my team encouraged to remove TD?**
  - Whenever they/we want
  - When they/we have extra time, budget, or human resources to be allocated
  - When they/we have a specific amount of time dedicated to TD removal (e.g., 10, 20%, etc.)
  - When they/we provide a business case for removing TD (e.g., reporting on costs, risks, and benefits of removing or keeping TD)

References

[1] E. Tom, A. Aurum, R. Vidgen, An exploration of technical debt, J. Syst. Softw. 86 (6) (2013) 1498–1516.
[2] H. Ghanbari, T. Besker, A. Martini, J. Bosch, Looking for peace of mind? Manage your (Technical) Debt - An exploratory field study, in: Proceedings of the 11th International Symposium On Empirical Engineering and Measurement, ESEM, 2017.
[3] Z. Li, P. Avgeriou, P. Liang, A systematic mapping study on technical debt and its management, J. Syst. Softw. 101 (2015) 193–220.
[4] T. Besker, A. Martini, J. Bosch, Time to pay up - technical debt from a software quality perspective, in: Proceedings of the 20th Ibero American Conference on Software Engineering (CibSE) @ ICSE17, 2017.
[5] W. Cunningham, The WyCash portfolio management system, in: Proceedings of the 7th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '92, 1992, pp. 29–30.
[6] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, N. Zazworka, Managing technical debt in software-reliant systems, in: Proceedings of the FSE/SDP workshop on Future of software engineering research, 2010, pp. 47–52.
[7] T. Besker, A. Martini, J. Bosch, Technical debt triage in backlog management, in: Proceedings of the Second International Conference on Technical Debt, Montreal, Quebec, Canada, 2019, pp. 13–22.
[8] K. Elena, J.R. Evaristo, S. Mark, Levels of culture and individual behavior: an investigative perspective, J. Glob. Inf. Manag. (JGIM) 13 (2) (2005) 1–20.
[9] S.M.U Gneezy, P Rey-Biel, When and why incentives (don't) work to modify behavior, J. Econ. Perspect. 1261 (4) (2011) 191–210, 25.
[10] A.A. Chughtai, Linking affective commitment to supervisor to work outcomes, J. Manag. Psychol. 28 (6) (2013) 606–662.
[11] S. Beecham, N. Baddoo, T. Hall, H. Robinson, H. Sharp, Motivation in software engineering: a systematic literature review, Inf. Softw. Technol. 50 (9-10) (2008) 860–878.
[12] A.C.C. França, T.B. Gouveia, P.C.F. Santos, C.A. Santana, F.Q.B.d. Silva, Motivation in software engineering: A systematic review update, in: Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), 2011, pp. 154–163.
[13] P. Milne, Motivation, incentives and organisational culture, J. Knowl. Manag. 11 (2007) 28–38.
[14] F. Fagerholm, M. Ikonen, P. Kettunen, J. Münch, V. Roto, P. Abrahamsson, Performance alignment work: how software developers experience the continuous adaptation of team performance in Lean and Agile environments, Inf. Softw. Technol. 64 (2015) 132–147.
[15] R.H. Rasch, H.L. Tosi, Factors affecting software developers' performance: an integrated approach, MIS Q. 16 (3) (1992) 395–413.
[16] S. Bala, J. Mendling, Monitoring the software development process with process mining. Business Modeling and Software Design, Cham, 2018, pp. 432–442.
[17] M.J. Bateman, T.D. Ludwig, Managing distribution quality through an adapted incentive program with tiered goals and feedback, J. Organ. Behav. Manag. 23 (1) (2004) 33–55.
[18] F.S.F. Soares, G.S.d.A. Junior, S.R.d.L. Meira, Incentive systems in software organizations, in: Proceedings of the 2009 Fourth International Conference on Software Engineering Advances, 2009, pp. 93–99.
[19] R.A. Ayala, Thinking of conceptual reviews and systematic reviews, Nurs. Inq. 25 (4) (2018) e12264.
[20] J. Hulland, Conceptual review papers: revisiting existing research to develop and refine theory, AMS Rev. 10 (1) (2020) 27–35.
[21] C.A. Ramus, Encouraging innovative environmental actions: what companies and managers must do, J. World Bus. 37 (2) (2002) 151–164.
[22] W. Snipes, V. Augustine, A.R. Nair, E. Murphy-Hill, Towards recognizing and rewarding efficient developer work patterns, in: Proceedings of the 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 1277–1280.
[23] Y. Wang, M. Zhang, Penalty policies in professional software development practice: a multi-method field study, in: Proceedings of the 2010 ACM/IEEE 32nd International Conference on Software Engineering, 2010, pp. 39–47.
[24] C. Seaman, Y. Guo, Chapter 2 - Measuring and monitoring technical debt, in: M. V. Zelkowitz (Ed.), Advances in Computers, Elsevier, 2011, pp. 25–46.
[25] T. Besker, A. Martini, J. Bosch, Software developer productivity loss due to technical debt—A replication and extension study examining developers' development work, J. Syst. Softw. 156 (2019) 41–61.
[26] T. Besker, A. Martini, J. Bosch, How regulations of safety-critical software affect technical debt, in: Proceedings of the 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2019, pp. 74–81.
[27] T. Besker, A. Martini, R.E. Lokuge, K. Blincoe, J. Bosch, Embracing technical debt, from a startup company perspective, in: Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018, pp. 415–425.
[28] S. Freire, N. Rios, M. Mendonça, D. Falessi, C. Seaman, C.O. Izurieta, R. Spínola, Actions and impediments for technical debt prevention: results from a global family of industrial surveys, in: Proceedings of the 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, 2020, pp. 1548–1555.
[29] P. Kruchten, R.L. Nord, I. Ozkaya, Technical debt: from metaphor to theory and practice, Software 29 (6) (2012) 18–21.
[30] T. Besker, A. Martini, J. Bosch, Technical debt cripples software developer productivity - a longitudinal study on developers' daily software development work, in: Proceedings of the First International Conference on Technical Debt @ ICSE18, 2018.
[31] S. Easterbrook, J. Singer, M.-A. Storey, D. Damian, Selecting empirical methods for software engineering research, in: F. Shull, J. Singer, D.I.K. Sjøberg (Eds.), Selecting empirical methods for software engineering research, Guide to Advanced Empirical Software Engineering (2008) 285–311.
[32] T. Punter, M. Ciolkowski, B. Freimut, I. John, Conducting on-line surveys in software engineering, in: Proceedings of the 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings., 2003, pp. 80–88.
[33] R. Czaja, J. Blair, Designing Surveys: a Guide to Decisions and Procedures, Pine Forge Press, Thousand Oaks, Calif, 2005.
[34] V. Díaz de Rada, Peter V. MARSDEN y James D. WRIGHT Handbook of Survey Research, 2010, Revista Internacional de Sociología 71 (1) (2013) 229–233.
[35] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2009) 131–164.
[36] R.A. Krueger, M.A. Casey, Focus Groups: a Practical Guide for Applied Research, Sage Publications, Thousand Oaks, Calif, 2009.
[37] V. Braun, V. Clarke, Using thematic analysis in psychology, Qual. Res. Psychol. 3 (2) (2006) 77–101.
[38] M. Vaismoradi, H. Turunen, T. Bondas, Content analysis and thematic analysis: implications for conducting a qualitative descriptive study, Nurs. Health Sci. 15 (3) (2013) 398–405.
[39] V. Braun, V. Clarke, Using thematic analysis in psychology, Qual. Res. Psychol. 3 (2) (2006) 77–101.
[40] J.L. Campbell, C. Quincy, J. Osserman, O.K. Pedersen, Coding in-depth semistructured interviews problems of unitization and intercoder reliability and agreement, Sociol. Methods Res. (2013).
[41] T. Besker, A. Martini, J. Bosch, Carrot and stick approaches when managing technical debt, in: Proceedings of the Third International Conference on Technical Debt, Seoul, Republic of Korea, 2020, pp. 21–30.
[42] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2) (2009) 131–164.
[43] R. Heiberger, N. Robbins, Design of diverging stacked bar charts for Likert scales and other applications, J. Stat. Softw. 57 (issue 5) (2014) 1–32. Pages.
[44] R.Howorko Indratmo, J.M. Boedianto, B. Daniel, The efficacy of stacked bar charts in supporting single-attribute and overall-attribute comparisons, Vis. Inform. 2 (Issue 3) (2018) 155–165. Pages.
[45] M. Streit, N. Gehlenborg, Bar charts and box plots, Nat. Methods 11 (117) (2014).