Journal of Software: Evolution and Process

WILEY

Journal of Software: Evolution and Process

RESEARCH ARTICLE - EMPIRICAL

Process Debt: Definition, Risks, and Management

Antonio Martini[1] | Viktoria Stray[1,2] | Terese Besker[3] | Nils Brede Moe[2] | Jan Bosch[4]

[1]University of Oslo, Oslo, Norway | [2]SINTEF, Trondheim, Norway | [3]RISE Research Institutes of Sweden, Gothenburg, Sweden | [4]Chalmers University of Technology, Gothenburg, Sweden

Correspondence: Antonio Martini (antonima@ifi.uio.no)

Received: 31 January 2024 | **Revised:** 28 January 2025 | **Accepted:** 15 March 2025

Funding: This work was supported by the Research Council of Norway (321477 and 340991) and Software Center (industrial consortium in Sweden).

Keywords: process debt | qualitative study | software development | software process improvement

ABSTRACT

Process debt, like technical debt, can be a source of short-term benefits but often leads to harmful consequences in the long term for a software organization. Despite its impact, the phenomenon of process debt has not been thoroughly explored in current literature, leaving a gap in understanding how it affects and is managed within organizations. This paper addresses this gap by defining process debt, describing its occurrence, the risks of its mismanagement, and showing examples of mitigation strategies. Our study began with an exploratory phase involving semi-structured interviews with sixteen practitioners across four international organizations, allowing us to gather diverse insights into the occurrence and management of process debt. Then, to deepen our understanding and validate our findings, we conducted a cross-company focus group with ten additional practitioners and analyzed fifty-eight observations and thirty-five interviews from a longitudinal case study. The analysis of the research findings led to a definition of process debt and a novel framework. We also report on the causes, consequences, and occurrence patterns of process debt over time. We present mitigation strategies and discuss which ones need further attention for future research. Our results suggest that the debt metaphor may help companies understand how to manage and improve their processes and make process-related decisions that are beneficial both in the short and long term.

1 | Introduction

Software process improvement (SPI) is a widely studied area in software engineering. Finding systematic and efficient ways to produce software that effectively delivers business value to customers is one of the most important goals for software organizations. Clear examples have been the efforts in designing and adopting new processes, from the Waterfall model to the V model to the most recent agile trend [1]. Many new processes and activities are continuously proposed as the field evolves and new domains and products emerge. Software organizations need to design and tailor new and efficient processes continuously to guide their teams [2]. Consequently, several strategies and maturity models to assess software processes have been proposed (e.g., Six Sigma [3], CMMI [4]). Choosing a suboptimal process can lead to disastrous consequences. For example, conflicting or not well-synchronized processes might put a large organization at a standstill, resulting in wasted time or even the lack of key documents for critical software such as certifications, jeopardizing the safety of the users. Several factors contribute to designing optimal processes or assessing improvement needs [5].

Nevertheless, what happens if the processes are not followed or are not well designed? In other words, how do organizations manage suboptimal processes? Research and industry often provide new ideas on how to implement new processes, while knowledge of change management can assist in implementing them [6]. However, important questions need to be addressed, such as how to best prioritize the improvements of the

© 2025 John Wiley & Sons Ltd.

processes that are already in place and how one may combine such improvement work with other pressing activities, such as delivering features to the customers and fixing bugs. Is process improvement perhaps down-prioritized in favor of such key activities? In other words, are companies accumulating Process Debt (PD), and if so, how do they manage such debt?

Recent research has brought quite a lot of new knowledge on the Technical Debt (TD) phenomenon to light, which characterizes suboptimal technical implementations that give a short-term benefit but create a context where a long-term interest is paid. For example, we know that TD is very harmful and can account, on average, for a hidden 30% waste of development time in companies [7]. In addition, it negatively affects developers' morale and software quality [8]. However, we do not have much information about PD. Is PD also similarly harmful, and how and why does PD occur? PD is a fundamentally different phenomenon from TD. Although in some preliminary papers, PD was included as TD, the definition of TD achieved during Dagstuhl has remarked that PD is not part of TD (although some issues might be connected to each other).

Today, only a few studies (briefly) mention PD, e.g., the study by Li et al. [9]. In our recent study [10], when analyzing the topics discussed in software development teams' agile retrospectives, it appeared clear that, besides TD, developers discussed several other non-technical issues that were deemed important for their performance. About one-third of the reported items were related to suboptimal processes that needed to be fixed. In other words, they reported instances of PD that had a direct negative consequence on the developers and their work. Such a study provides a preliminary definition, but without a full empirical investigation to support it. In summary, PD, similarly to TD, is potentially a harmful phenomenon affecting software practitioners' performance and the quality of software. Despite this, there is a knowledge gap about PD.

In this paper, we aim to address this gap by exploring PD through an empirical study by gathering experiences from practitioners in four international software companies. In particular, we investigated the following research questions:

- RQ1: What is PD?
  - RQ1.1: How can PD be defined?
  - RQ1.2: What types of PD exist?
- RQ2: How does PD occur?
  - RQ2.1: How is PD accumulated?
  - RQ2.2: What are the causes of PD?
  - RQ2.3: What causes are the most common?
  - RQ2.4: What are the negative consequences of PD?
  - RQ2.5: What consequences are the most harmful?
- RQ3: How can PD be managed?
  - RQ3.1: What mitigation strategies do companies use to manage PD?
  - RQ3.2: What mitigation strategies are still missing to manage PD?
- RQ4: How did PD change over time, and how was it managed?

The main contributions are:

- A comprehensive framework to practically and theoretically reason about PD, including:
  - causes, consequences, and types of PD
  - occurrence patterns and evolution of PD over time
- A survey of the state of practice for PD management in five companies
  - evidence of concrete instances covering each type of PD
  - mitigation strategies to manage PD
- Extensive validation of results by triangulating methods and sources across contexts

The remainder of this paper is organized as follows. First, we outline existing key concepts, for example, related to SPI. Then we present our methodology, and we report the results in the same order as the RQs. Then we discuss the results, limitations, and related work, and we conclude with our final remarks.

2 | Related Work

2.1 | PD and TD

TD research has increased steadily in the last 10 years, showing great interest both from academia and the industry [9]. However, the TD metaphor is mainly restricted to enclose only technical issues, such as those related to code, tests, architecture, and documentation [11]. Other issues, such as social debt [12], were excluded from the set of TD issues and considered as different kinds of debt. In addition, issues concerning builds, infrastructures, etc., which were previously considered TD (as in the systematic mapping [9]), were also removed from the TD scope. The main reasons were not related to the applicability of the debt metaphor but rather to the substantial difference in managing issues related to technical artifacts with respect to non-technical aspects. For example, tools, methods, and strategies to manage social debt, where the relationships among humans are involved, can differ significantly from static code analyzers used to assess code. In summary, while TD has become well defined and has a community to study it (e.g., the International Conference on Technical Debt), several issues, also deserving the "debt" status, seem to have been left behind. At the same time, new kinds of debts emerge continuously in practice as important issues to be addressed [13], as testimony that the debt metaphor can help to reason about other aspects than the strict technical ones.

Our study used the only existing definition, given in Martini et al. [10]: PD is "a suboptimal activity or process that might have short-term benefits but generates a negative impact in the medium-long term." However, such a definition is based on TD's definition. Therefore, we also aim to develop a more nuanced and comprehensive definition based on empirical evidence in this study. As with TD [14], we assume that the PD metaphor is composed of similar components, namely, the debt, the interest, and the principal. A debt item consists of the divergence, in practice, from an optimal process; the interest constitutes several negative effects generated by the occurrence of such debt; the principal corresponds to the cost of changing the process to avoid or repay the debt.

Although a process is, most of the time, the result of a tradeoff across several stakeholders' needs, some tradeoffs might be better or worse than others. In this case, we do not recognize the debt as a specific suboptimality (which can still be acceptable in the best possible tradeoff), but by assuming that a better tradeoff (and therefore a better process) could be achieved to solve the same problem. However, a suboptimal process is not considered PD if it does not have a clear interest to be paid, as it has been postulated for the TD metaphor.

2.2 | Process and Workflow

A process is a structured set of activities and decisions to do a certain job.

> A software development process, also known as a software development lifecycle, is a structure imposed on the development of a software product. A software process is represented as a set of work phases that is applied to design and build a software product. There is no ideal software process that suits all different types of software developing companies, and many organizations have developed or tailored their own approach to software development.
>
> [15]

In practice, there might be several subprocesses that are interconnected. The process can emerge or be designed. There can be one or more process designers (those who are in charge of designing and managing the process) and one or more process executors (those who perform one or more steps in the process). In some cases, these roles might coincide. In agile software development, software process initiatives are often built into retrospectives [16].

The difference between workflow and process is that a workflow is a series of repeatable activities that one needs to carry out to finish a task. A process, on the other hand, is a set of repeatable activities that need to be carried out to accomplish some organizational goals [17]. In practice, the workflow represents the everyday sequence of activities from an individual point of view. In contrast, a process represents an overall sequence of activities carried out by specific roles to reach an overall goal. Different processes (software related or not) might affect different aspects of software development: for example, a process can be defined to implement a testing strategy or to produce documents for certification of the software with respect to safety, security, other requirements, etc. Other processes, for example, in mechatronic products, might be used to manage other parts of the product but can still affect software development.

2.3 | SPI and PD

In software development, there is a long tradition of regularly improving the processes [18]. SPI is about making the software processes better and is mainly a human activity [19]. SPI has been found to be positively associated with business success in small to medium-sized companies [20]. However, process problems are frequently identified but rarely solved [21], and then, therefore, only the potential for improvement exists.

One important driving force for SPI initiatives is that the team members learn how to improve their activities. SPI can be seen as an organizational change mechanism, which requires group learning [22] because collaboration is essential in the software development process. Agile software development supports group learning through frequent feedback sessions like stand-up meetings, demos, and retrospectives involving the whole team.

In their learning theory, Argyris and Schon [23] distinguish between two types of learning in organizations: "single-loop learning" and "double-loop learning." Single-loop learning refers to changing processes in response to a problem to prevent its recurrence. Examples include managers monitoring development costs, sales, or customer satisfaction to ensure organizational activities remain within predetermined limits. In this approach, actions are slightly modified to achieve desired results when expectations are not met. A feedback loop connects observed effects with refinements to influence outcomes. Double-loop learning, on the other hand, examines the underlying factors that influence these effects and is called the governing values [23]. It involves using the problems being experienced to understand their root causes and then taking actions to address these causes. For instance, correcting a software error is single-loop learning, but addressing the cause of the error is double-loop learning. The changes based on this type of understanding will be more thorough. To sum up, single-loop learning is about asking, "Are we doing things right?" while double-loop learning is about asking, "Are we doing the right things?"

If organizations focus only on single-loop learning, the underlying cause of PD will not be solved, and the suboptimal process will continue to exist. Therefore, we argue that to reduce PD, companies need to conduct double-loop learning. That is, the team or the organization needs to "learn how to learn," also known as deuterolearning [23]. Teams need both to conduct single-loop learning (questioning if they are doing the process right) and double-loop learning (questioning if they are doing the right process). In addition, they need to make the right decisions when replacing a suboptimal process with an optimal one in order to reduce PD.

Although there exists a large body of knowledge related to SPI and change management, our goal for this study is to understand what PD is and how it is accumulated and managed, which has not been studied before. The results will provide companies with new knowledge on how to conduct double-loop learning and better understand their PD. To do so, and to maintain an exploratory approach, we decided to conduct our investigation without using previous knowledge related to SPI to formulate our questions, but rather to let the lens of PD direct our interviews in a fully exploratory fashion.

2.4 | Extension to Our Previous Work

This study was partly and originally published at the APSEC 2020 [24]. The delta of this manuscript over the prior published

study is shown in Table 1 below. Some data from the longitudinal study (19 interviews, 39 observations) has earlier been used to study other phenomena such as digital transformation [25] and teamwork and coordination [26]. Some details and quotes have been removed for the sake of length and can be found in the previous paper [24].

3 | Methodology

As visualized in Figure 1, this study's research design was divided into three main steps, including six different phases. The first three phases address an exploratory case study, which was conducted in Step 1 (Phases 1–3) and primarily refer to this publication's original study [24]. The following two steps were conducted as an extension to that study and included a longitudinal case study (Step 2, Phase 4) and also a final cross-company focus study (Step 3, Phases 5 and 6). Figure 1 also describes what activities were performed in each of the phases, together with the research methods used in each phase.

3.1 | Step 1—Exploratory Case Study

The first step of this study's exploratory nature aims to answer RQ1, RQ2.1, RQ2.3, RQ2.4, and RQ3 and is described in the following subsections.

3.1.1 | Phase 1—Contextual Analysis and Design

Case studies have been long established in the software engineering field to explore how different processes and tasks are carried out in practice in the industry. Due to the exploratory nature of the study, we employed a multiple qualitative case study approach [30]. First, the study was presented and discussed during a workshop with one or more participants from each of the five involved companies. Such participants were always employees with management responsibilities related to the processes, methods, and tools (PMT) employed in the organizations. This phase acted both as an introduction session where each company briefly described their overall software development process together with an assessment of concerned stakeholders of the process.

The goal of these workshops was to introduce the participating companies to the study, to align and equip them with relevant knowledge about the concept of PD (we showed the definition, we explained the metaphor, and we reported a few anecdotal examples), and to gather background and contextual information on each participating company in preparation for the following interviews. Further, the aim of this stage was also to communicate and describe what the "process" term refers to within this study's context and also to identify potential interviewees for the next stages of the study. Each workshop lasted from 30 to 60 min and was digitally recorded.

TABLE 1 | Comparison of previous published work and this manuscript.

	Previous paper [24]	Added in this paper
RQs	RQ1, RQ1.1, RQ1.2, RQ2.1, RQ2.2, RQ2.4	RQ2.3, RQ2.5, RQ3, RQ3.1, RQ3.2, RQ4
Data	Workshop 4 companies 16 semi-structured interviews	Cross-company study with 4 companies Validation survey with 10 participants Longitudinal case study, 6 years long 19 interviews, 39 observations
Companies	Companies A–D	Companies A–E

3.1.2 | Phase 2—Qualitative Data Collection

The data collection method was a combination of interviews together with the analysis of internal company documents. This study employed a combination of unstructured and semi-structured interviews, where the questions were both formulated as general concerns and interests from the researcher about PD.

The interviews aimed to explore the concept of PD within each of the companies and what aspects impact and drive such debt (see Table 2). We started by asking the interviewees to describe their overall software development process, both from a historical and a systematic perspective. We asked four follow-up questions to learn about the aspects of the process, focusing on the characteristics described in Section 3.1.1 (according to our RQs).

Further, to get more insight into the existing PD, the interviewees were also invited to share their internal documents describing their process and to share their experience with a specific PD task that was identified before the interview took place. However, due to confidentiality reasons, these documents are not publishable.

All interviewees were asked for recording permission before starting, and all interviews agreed to be recorded and to be anonymously quoted for this paper. Each interview lasted between 60 and 120 min and was transcribed verbatim. To improve the reliability of the collected data, at least two of the authors participated in each interview session.

Taken together, this study includes seven focus interviews with 16 practitioners from four companies, where each interview included between 1 and 6 participants (according to the participants' preferences). Given the exploratory purpose, the interaction among participants was important to understand the possible dynamics of how PD was accumulated. For confidentiality, interviewees and their companies are kept anonymous.

3.1.2.1 | Cases Description. To provide more context for our study, this section describes the involved participants

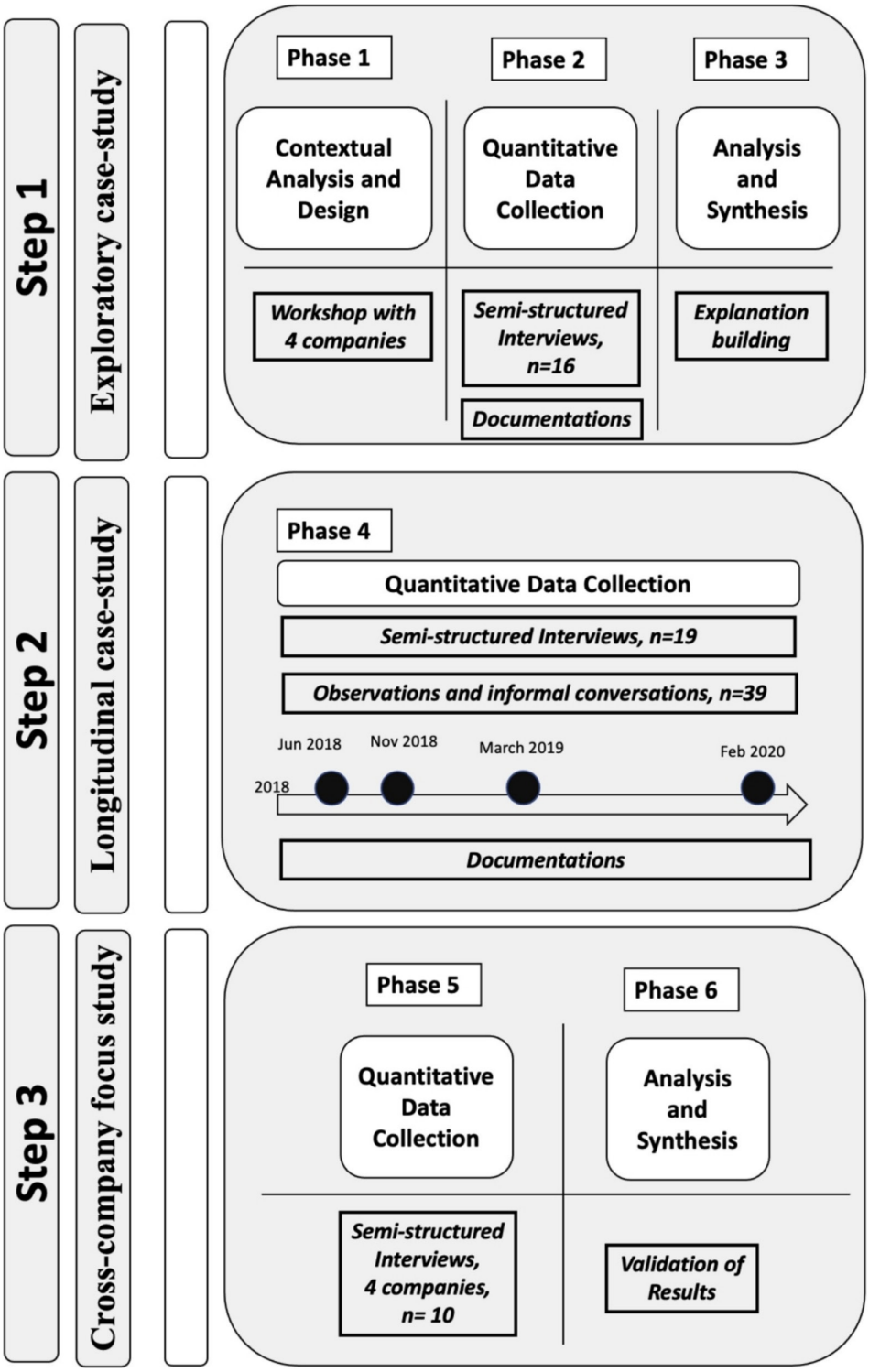

FIGURE 1 | Visualization of the research model and method used in each phase.

in more detail. All case companies have an extensive range of software development activities, specializing in different business and application domains. Company A develops software applications used within the car manufacturing industry. Company B develops software for water processing applications. Company C develops software used in applications such as, e.g., cars and IoT types of equipment. Company D develops software applications used in the energy and power production section.

3.1.2.2 | **Participants.** As illustrated in Table 3, the interviewees work at companies operating in several different business domains, and they have several roles.

3.1.3 | Phase 3—Analysis and Synthesis

Based on recommendations from Yin [27], the analysis was carried out using a code and pattern identification technique called

TABLE 2 | Questions asked in the first step of the study.

Question	Relation to RQ
Describe some critical suboptimal processes	RQ1
What led the process to become suboptimal?	RQ2.1, RQ2.2
What are the consequences of such suboptimality?	RQ2.4
How do you prioritize improving the process with respect to other software development activities?	RQ3
How useful would it be to reason about these issues using the PD metaphor?	RQ1

TABLE 3 | Participants in the first step of the study.

Company	Participant roles
A	Manager (testing), manager (process, methods, and tools), software architect
B	Two managers (process, methods, and tools), senior software developer (requirements)
C	One manager (process, methods, and tools), process expert, software process manager, software architect, senior software developer (continuous integration), software architect manager (methodology)
D	Senior software improvement manager, technical manager, team leader, senior software developer

TABLE 4 | Participants for the second step of the study.

Company	Program	Participant roles
E	Web	Project manager, manager, business developer, digital responsible, digital designer, team lead, tech lead, test lead, IT developer
	Agile	Product owner, four developers, test leader, two business developers, enterprise architect, UX designer

explanation building to match the analysis from both the interviews and the documents into one because this technique is specifically suitable for exploratory studies. This selected analysis technique assisted us in explaining the PD phenomenon by stipulating a set of relational links related to PD, addressing aspects such as "how" and "why" PD occurs [27]. The technique of gradually and iteratively building, comparing, grouping, and revising findings among the different cases resulted in a final exploration of PD.

3.2 | Longitudinal Case Study

3.2.1 | Phase 4—Qualitative Data Collection

Company E has in-house software development units in Sweden and Norway. We have closely followed the case from 2014 until 2020. The main reason for choosing this case was that the company was part of two research projects on process improvement. They had implemented a build-measure-learn methodology on their processes, which made it possible for us to observe suboptimal processes over time and to understand improvement initiatives and their effects. We followed two separate programs: one program with three teams called the Web program and one with six teams called the Agile program. The data was collected between June 2014 and September 2020. An analysis of the interviews conducted in the Web Program regarding the company's agile digital transformation is reported in Mikalsen et al. [25]. An analysis of the interviews conducted in the Agile program regarding teamwork and coordination in distributed development is reported in Stray et al. [26].

3.2.1.1 | Web Program. As we entered the case at the beginning of 2014, the software development unit in Company E was organized in a hierarchical and modular structure. Within this structure, units were based upon the modules constituting their digital portfolios, for example, banking and insurance, and digital and mobile. In this period, Company E was deploying a new program that was transforming the way the software development unit delivered digital offerings in the bank. Instead of having technical modules as the central organizing concept, they moved towards a delivery model consisting of five delivery streams (e.g., insurance, banking, pension). The goal of the program was to implement a new delivery model for digital solutions. Effects the program sought included giving development clearer frames regarding resources (i.e., hours), a more unified prioritization of tasks, rapid delivery, stable team participation, a unified development method, and a predictable frequency for prioritized deliverables.

We conducted nine semi-structured interviews with participants in the Web program, as detailed in Table 4. Additionally, we collected various documents, including plans, strategy reports, progress reports, evaluations, sketches, and system designs. We observed a total of 24 meetings. We wanted to observe different kinds of meetings and chose to observe:

- 12 daily standup meetings
- 2 weekly meetings
- 1 planning meeting
- 7 general meetings
- 2 retrospective meetings

During most meetings, we made notes during the meeting; however, for the standup meetings, notes were taken immediately afterward. The two retrospective meetings involved more engaged research, meaning that we also assisted the participants in solving their difficulties with practical and theoretically grounded solutions. In these meetings, we had one observer taking notes.

3.2.1.2 | Agile Program. In 2017, Company E initiated an agile program that consisted of four cross-functional, autonomous teams. These teams were organized according to agile principles and tasked with developing software for business-to-business insurance products. The teams included personnel from both the software development and business development departments and were focused on delivering software solutions to the business side of the organization, such as sales and settlements. They also worked closely with the organizational departments responsible for innovation and technology development.

We conducted ten interviews with participants in the Agile program. These interviews lasted from 37 min to an hour, with an average of 48 min. All interviews were recorded with consent and transcribed verbatim within a week of the final interview, which took place in January 2019. Additionally, we facilitated three workshops and observed 15 meetings, including 11 daily stand-up meetings and four weekly progress meetings. We also collected various documents, such as team presentations, progress plans, and analysis results from internal surveys. The documentation was useful for gaining a better understanding of the context, and it also helped verify specific details.

3.2.1.3 | Data Collection and Analysis—Company E. The interviews aimed to explore suboptimal processes for working agile, for meetings, information flow, and coordination. We investigated what could work better in the project and how they solved the problems. We also asked how they perceived the code quality and how this could be improved. We did not specifically use the metaphor of PD but instead asked about suboptimal processes and their consequences.

Our data analysis was guided by an interpretive approach that places practitioners' understanding of reality at the core of our study [28]. We subscribe to the concepts proposed by Klein and Myers [29], of which the hermeneutic circle is a central principle. The hermeneutic circle helps to account for the interconnected meaning of the parts (e.g., the understanding of the participants) and the whole that they form (e.g., the meanings emerging from the interactions between the parts). In our data analysis and sensemaking technique, we adhere to this idea using an inductive–deductive approach. Our findings highlight the interdependencies and PDs that exist in Company E, such as synchronization debts, PDs resulting from interdependencies, and varied and inappropriate procedures.

3.2.2 | Step 3—Cross-Company Focus Study

In order to validate our results and to gain additional insights from the combined discussion across multiple companies, we conducted a group interview where we reported our results and followed up with validatory questions as well as additional exploratory questions.

The participants in the group interviews were a mix of previous interviewees and additional process improvement experts from the companies involved in Phase 1. One of the purposes was to validate the elaborate models and findings from Phases 1 and 2.

In total, we involved 10 interviewees from four companies, while three researchers presented the results and asked additional questions. The total interview time was 3 h.

The group interview was structured as follows. First, we had an introduction of the participants and collected demographic information. Then, we reported and explained the overall framework to define PD. During our explanation, we gave the participants room to ask questions, clarify issues with the model, and suggest changes. During this part, we also presented the types of PD revealed by our analysis. The purpose was to validate our initial framework related to RQ1.

Then, we alternated the presentation of results and collection of feedback on them, combining open discussions driven by the participants with the anonymous and individual collection of closed answers using the instant survey instrument Mentimeter. We then showed the results live during the workshop and collected additional feedback. In particular, this part was divided into the following three subparts:

1. **Causes of PD:** We showed our categorization of the causes, and after discussion, we asked the question, "pick the three causes that generate the most PD in your organization." This way, we also managed to provide a ranking of which causes seem to be the most common and if there are differences across contexts (RQ2.3).
2. **Mitigation strategies:** We showed our categorization of the mitigation strategies for PD, and we asked the question, "pick the three mitigation strategies that you are currently using the most in your organization." Then we also asked, "pick the three mitigation strategies that you would like to apply at your organization."

   These two questions were asked to investigate the difference between the current state of practice and the wanted position by the companies (also useful to understand which mitigation strategies should be prioritized by research) (RQ3.2).
3. **Consequences of PD:** We showed our categorization of the causes for PD, and we asked the question, "pick the three consequences that have the most impact on your organization." This question was asked to rank the most harmful PDs (RQ2.5).

4 | Results

Here, we present the results of our investigation. First, we outline the overall framework and definition to answer RQ1; then we answer RQ2 and RQ3 by reporting the initial results from the interview in Step 1, followed by the additional validation results from Step 2 (the longitudinal case) and Step 3 (the cross-company interview). We also add, for RQ2.3, RQ2.5, and RQ3, the results from the questionnaire organized during the data collection of Step 3. Finally, we report the results from RQ4 related to the longitudinal case study.

4.1 | PD Definition (RQ1.1)

We propose an overall PD conceptual framework (Figure 2) refined from the one in Martini et al. [24], where all the key elements and their relationships are visible (RQ1). This is a high-level model, and additional details about the entities and connections are explained in our findings according to the various RQs.

An *optimal process* is either designed or has emerged in the organization, usually depending on their culture and practices (some companies have a top-down, usually more formal, approach to defining processes, while others have a more bottom-up approach, where processes emerge from the needs of the stakeholders). Optimal processes are usually defined in terms of an optimal tradeoff among stakeholders.

We define an optimal process and tradeoff here as the one that gives the most value to the involved stakeholders. However, it is likely that, in practice, the optimal process recognized by an organization would still not be theoretically optimal but rather the best option recognizable by the organization. When we refer to an optimal process in the rest of this paper, we refer to this latter one and not a theoretical one. For example, a company working with a perfectly functioning waterfall approach (in theory) might have debt if they could identify that Agile would be superior in their context.

An optimal process can be subject to PD and its accumulation according to three high-level accumulation patterns; in other words, there are three main distinguishable ways in which PD usually occurs. The presence of PD generates consequences (often in terms of negative effects) that affect the stakeholders, balancing the tradeoff and making the process suboptimal. Consequences can be short-term or long-term: in the first case, PD generates an initial waste, which can be reduced and mitigated but can also develop into long-term consequences.

Several causes trigger the PD accumulation patterns to happen (will be discussed in Section 4.6). The starting point responsible for the accumulation of PD is either a process designer (who has created the process with a suboptimal design) or a process executor (who has diverged from the decided process). In addition, the main source of PD can be a suboptimal infrastructure. When a process emerges, the process designer and executor can often be the same, and in such cases, PD often occurs because of an involuntary suboptimal design.

PD can be identified by a type: this can be a suboptimal aspect of the process (role, documentation, synchronization with other processes, etc.) or by a specific suboptimal activity (this can depend on the specific activities that the process is composed of, for example, is it a code review step and prioritization). A combination of aspects and activities can also be used, for example, "lack of prioritization roles."

The negative effects of PD can be reduced by applying mitigation strategies. These can be applied by developers and managers, but such mitigation strategies are costly and need to be prioritized based on some key factors.

To prioritize mitigation strategies for PD, one needs to assess and estimate the interest (the cost of negative consequences), the principal (the cost of changing the process), and the value for the stakeholders. When promoting a change to mitigate a specific PD, organizations need to take into consideration these three variables and weigh them, taking into consideration the costs and benefits of applying such a mitigation strategy with respect to implementing other strategies instead or just continuing to develop features.

For example, it is possible that changing a prioritization process would be costly but would also mitigate several negative consequences, while changing a code review process for a specific team would not cost much and would give limited benefits. Often managers and developers need to prioritize either of the two (for example, because of project resource constraints). Let us assume that the prioritization process could give a much higher value to the stakeholders than the review process; the former one should be prioritized.

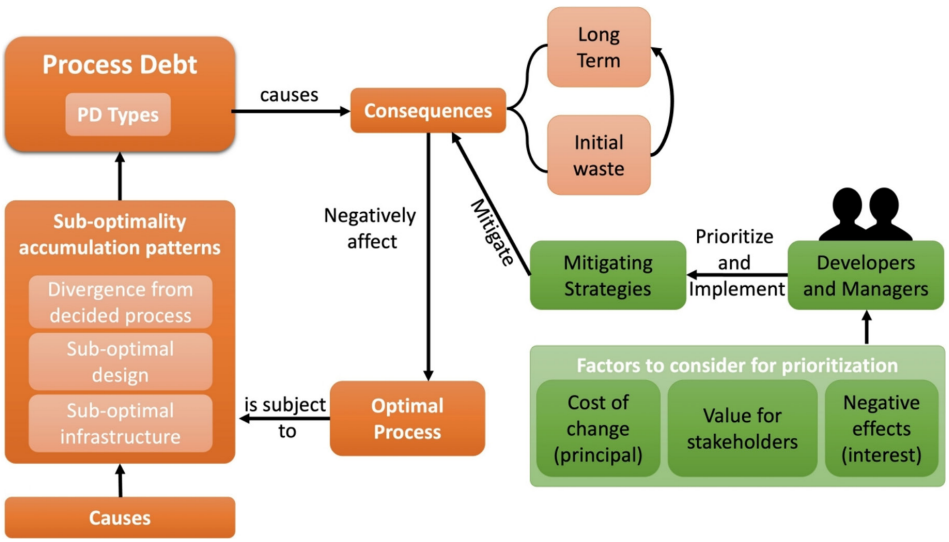

FIGURE 2 | The overall framework of PD (orange) and mitigation strategies (green).

It is important to notice that the prioritization of mitigation strategies also depends on other factors that are not PD-specific but can be constraints of a specific project, such as resources and time span. Such factors are already well-known from the literature on SPI and are often context-specific, so we do not report them in our framework for the sake of simplicity, but they would need to be taken into consideration when PD is assessed.

4.1.1 | Validation of the definition and framework

During the analysis of the longitudinal case study, we did not find additional insights to change the framework; the findings were supported by the gathered data.

The discussion with the practitioners during the cross-company interview in Step 3 showed that the practitioners appreciated having a framework to discuss such a complex problem, so the framework was validated. The input from the participants in the workshop, combined with the input from the longitudinal study and the addition of the mitigation strategies, led to the final framework in Figure 2, which has been expanded, refined, and simplified with respect to our previous publication.

Then, given that the PD definition proposed in Martini et al. [10] was based on the definition given for TD and we found several distinct differences between PD and TD, we propose the following revised definition for PD (RQ1):

> Process debt is the occurrence of a sub-optimal process design, divergence from an optimal process or deficiencies in the infrastructure that might be beneficial in the short term but might generate a long-term negative impact for process stakeholders.

4.2 | Types of PD (RQ1.2)

Besides categorizing PD based on where it originates, it is critical to understand what parts of the process are affected by the debt. Processes are made of several components, namely, roles, activities, and documentation. We provide a categorization based on such components in a similar way as what is commonly used to categorize TD (e.g., code debt, architecture debt, and requirement debt), with concrete examples from our cases. Based on the results from our analysis, six types were identified, as shown in Figure 3. We give examples found in the longitudinal study in Table 5.

4.2.1 | Activity-Specific Debt

Processes are created for different goals and may involve different activities. It is possible that one specific activity is suboptimal and not the whole process as a whole.

We found several kinds of activities that can be flawed: for example, a suboptimal prioritization process could be called prioritization debt, a suboptimal certification process could be called certification debt, and so on. There may be many activities that could generate debts. In addition, many activities are context-dependent, meaning that they appear in some companies and not in others, or they appear with different names. These all fall in the PD category.

This sort of debt is not mutually exclusive with respect to the other presented categories. The purpose of this category is to give the opportunity to better define a specific PD. For example, the unsuitable process identified in Company E during the longitudinal case study was about business decision-making. In this case, we would call this instance of PD an unsuitable (business) decision-making process.

In those cases where it is not only one activity being suboptimal, for example, the overall waterfall-like process for Company A, one would just refer to the whole process, such as the "Unsuitable Mechatronic Development Process."

4.2.2 | Mismatching Roles and Responsibilities (Roles Debt)

Activities are carried out by executors, who usually have different roles according to the responsibilities mapped in the process. Roles are not physical people but are "hats" that are dressed by anyone in the organization to carry out a specific activity. Further, often roles within the process are also associated directly with a role that is represented in the overall company organization. This can create a mismatch between the responsibilities described in the process and the responsibilities in the organizational structure, which might prevent employees from

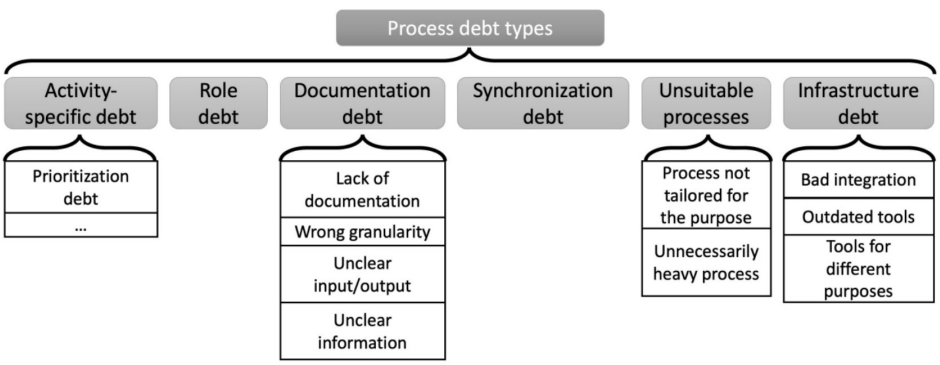

FIGURE 3 | Types and subtypes of PD.

TABLE 5 | Examples of process debt and their types from the longitudinal case study (Company E).

Example	Type of process debt
A role was created called product owner (PO), and people were assigned to that role without saying what the role entailed. Many of the people assigned to that role were confused as some were familiar with the product owner role in Scrum. However, the responsibilities of the product owner in Company E were more related to being a team leader.	Roles debt
Developers were not changing the status of their tasks themselves and did not assign them to the next designated role, so the PO had to do it for them, causing extra effort for the PO. The cause mentioned here was that the process was not well communicated in advance to the actors.	Roles debt
The offsite teams followed Scrum, while the onsite teams used Kanban. The different methods chosen affected how pull-requests were handled between the teams. Scrum teams sent a large number of pull requests to the onsite teams after the sprint, overwhelming them. Reviewing extensive pull requests became demotivating and time-consuming.	Synchronization debt
The business development unit followed a decision-making process that was not suitable for software development. The process involved complex decision-making with continuous synchronizations and updates, causing tasks planned in the sprint to be unsuitable and unpredictable.	Process unsuitability
People were forced to sit at specific desks two times a week as they belonged to two teams. This frustrated employees who wanted to sit in the same spot all week due to preferences for their team proximity or reliance on external monitors.	Infrastructure debt
Processes regarding using Jira caused infrastructure debt in Company E. Offshore teams were measured by the number of solved Jira issues, encouraging developers to attach multiple Jira issues to one pull request. Norwegian developers had to review very large pull requests, which was frustrating and inefficient compared with smaller, more frequent reviews.	Infrastructure debt
The company decided that the Norwegian teams would use Kanban while the outsourcing teams followed Scrum. Although the teams used processes they were familiar with, this caused synchronization issues, misunderstandings, and overhead. Coordination meetings became inefficient and caused a waste of time.	Suboptimal process design

accepting and carrying out some tasks for which they are responsible in the process.

4.2.3 | (Process) Documentation Debt

The process is usually described by documentation. Although, in some cases, for example, when only a few stakeholders are involved, the process can be kept informal; meanwhile, in large projects where more teams and stakeholders are involved, the process needs to be well documented. The lack of process documentation (or its inaccessibility) was recognized as one of the major sources of PD both by process designers and by process executors from across all studied cases, including the longitudinal case study.

Providing a suitable level of granularity for the process design is challenging. For example, the documentation might lack information about steps or stakeholders, which leads to unclarity and confusion, or, on the contrary, it might be too detailed. The latter case might seem somewhat counterintuitive: the

interviewees mentioned that having too many details could actually create as much overhead as having too few. For example, too many details might give wrong instructions on special cases that should be handled by the process executors (especially in the presence of frequent special cases). Another negative effect of such over-detailed documentation is that executors might skip part of it because they find irrelevant information for their specific needs. If the information is not well structured, an experienced practitioner might already be familiar with most of the steps and might find it difficult to locate the only important piece of information that is needed (e.g., an update to the process).

Besides the presence of information or the lack of it, it is also important that the documentation provides clear information. If the language and the graphical representation are not the ones that are used by the consumer of the documentation, this can have an impact on how the process is executed.

4.2.4 | Synchronization Debt

The stakeholders involved in a process are also likely to participate in multiple processes that are intertwined. For example, there might be a process such as Scrum to support agility, a separate process that prescribes code reviews to ensure software quality, and a third process to certify the delivered software according to some standard. The complexity can vary and might depend on the scale and domain of the organization. Such processes need to be synchronized via synchronization points. Suboptimal design of synchronization points may lead to steps being skipped, confusion, overhead, and disrupted workflow.

In general, stakeholders, besides being executors for a number of processes, also have a workflow. An example is a software developer who needs to perform a bug fix (a task that needs different steps) while at the same time, the code needs to be reviewed, certified, tested, etc. Different activities in the workflow of a development task might belong to different and parallel processes. One key finding from our interviews with the executors of the processes is that developers need to accommodate different external processes into their workflow. The interconnection of such processes might create a suboptimal workflow for the developers, where some activities conflict with each other. In conclusion, the main findings related to synchronization debt are that the lack of synchronization between processes has an impact on developers' productivity and that the lack of synchronization between processes and individual stakeholders' (especially developers') workflow can disrupt their workflow.

4.2.5 | Process Unsuitability

Some processes might not be suitable to support the business needs of an organization. The difference with synchronization debt is that, in the former case, the processes might be optimal but not well synchronized, whereas, for this category, there is one process that is not suitable for the needs of (a part of) the stakeholders. In summary, PD consists of the existence of a process, from which software development is dependent on, which creates overhead, confusion, and delays.

4.2.6 | Infrastructure Debt (ID)

IDs were considered to be PDs by the interviewees. For example, tools in the toolchain may work seamlessly with each other, or they may not. When tools that are part of the same process or are part of different processes but are interacting within the same workflow for the stakeholders and are not well integrated, they might create overhead, errors, or excessive task switching for their users, which qualifies as PD. Additionally, tools might be outdated and might become unfit to support modern processes.

In addition, there might not exist tools that fit well with the processes, or the right tools might require high costs. This might lead to tools that are not used for the purpose required by the process and might, therefore, be suboptimal. In the last case related to Company E, it is clear that not only is it the virtual environment and tools used to develop software that constitutes PD, but ID can also include the physical environment surrounding the practitioners.

4.3 | PD Occurrence and Its Accumulation Patterns (RQ2.1)

Three overall PD accumulation patterns were identified: suboptimal process design, process divergence, and tool deficiencies, as outlined below. PD is primarily initiated by two main actors, process designers and process executors, but we found that the infrastructure may also trigger the accumulation of PD. We call these three PD initiators.

4.3.1 | Suboptimal Process Design

This category includes those processes where a decision was made that caused the process to be suboptimal in some aspects and with respect to a number of stakeholders.

Although we call it "process design," our analysis distinguishes two cases: those where the process has formally been designed and those where the process has emerged over time without a clear upfront design. Although the strategies to manage these two cases might differ, from a conceptual point of view, there is no difference if the process was designed upfront or the design just emerged. In both cases, the process design would be flawed. In the interviews, several instances of this kind of PD were identified, two reported below.

4.3.2 | Process Divergence

This category refers to those processes that are initially effectively well-designed by the process designers but are not followed by process executors. Again, by well-designed here, we do not mean processes that do not have any suboptimality but for which a better tradeoff is not identified.

4.3.3 | Infrastructure Deficiencies

This category of PD contains those issues that are related to inefficiencies in the infrastructure supporting the processes. There are two subcategories: tools that are used to carry out processes (e.g., project management, bug tracking tools) and tools that are used to design the processes (e.g., to create guiding documents). PD exists if a (better) alternative infrastructure solution is known but is not in place.

As identified in Section 4.2.6, ID is part of PD. The infrastructure is there to support the process and does not have another independent function. This is why, usually, in organizations we find PMT units that need to coordinate processes and tools together. Given how intertwined the two categories are, it would not make much sense to separate ID from PD. In practice, any ID will also create a problem for the process. We refer to infrastructure deficiencies as ID (as part of PD) to denote that the tools are a source of the debt.

4.4 | Causes of PD (RQ2.2)

Our results show how a large number of causes can be responsible for the introduction and the presence of PD. The causes are listed in Table 6 and illustrated in Figure 4.

4.5 | Which PD Causes Are the Most Common? (RQ2.3)

Figure 5 shows how the participants from the cross-company interview chose the causes that most commonly lead to PD (RQ2.3). Three voters for Company A, two for Company B, three for Company C, and two for Company D.

The most voted cause, and the one that was voted on by all participating companies, is the presence of poor technology and tools. The second most voted cause, especially by Company A, was the neglect of process value. Special contexts and solutions, as well as a lack of improvement prioritization, were voted on by three organizations.

However, it is interesting to see how different roles and participants from different companies voted differently, suggesting that in different subcontexts, different causes might generate more PD. Overall, one could consider most of the causes as contributing to generating PD.

4.6 | Consequences of PD (RQ2.4)

When analyzing the consequences (Figure 6), it was evident that PD has two kinds of negative effects: besides the generation of a long-term interest (as for TD), process suboptimality can actually often lead to initial waste (of time, effort, budget, etc.) for the stakeholders. The initial waste can, in some cases, be reduced by refining the process. However, in many cases, such negative effects can also easily become a long-term interest. This means that the two groups are not mutually exclusive, but the consequences listed as initial waste might also belong to the long-term category, although they might show up early.

4.6.1 | Long-Term Consequences

1. Late customer deliveries: Although skipping some steps and activities in the process might promote quicker deliveries of customer value in the short term, the interviewees reported that this way of working could cost more in the long run and cause longer delivery times in the long term. In some cases, customers were unaware of the PD and its accumulation and consequences, which led to their dissatisfaction when unexpected long-term delays occurred.
2. Low software product quality and TD: One of the purposes of processes is to ensure that products delivered to customers meet several qualities critical for them. Many PD instances mentioned by the interviewees generate quality issues in the long term. For example, inadequate unit test coverage can create additional defects over time and undermine the organization's confidence in delivering additional features that are not appropriately tested (Company C). Additionally, some companies mentioned that PD often generates TD. For instance, suboptimal code reviews might not catch TD, while the lack of a process to manage iterative and continuous feedback loops between architects and development teams can lead to architecture debt.
3. Hindering other processes: Processes are often interconnected, and the presence of PD in some processes may cause disruption in others, generating new PD (e.g., in the case of synchronization debt). This interconnection can make PD contagious and may lead to vicious cycles. An example from our interviewees relates to an existing ID issue with an outdated tool whose technical limitations prevented the implementation of continuous integration.
4. Issues with certification: Companies A and D reported that skipping process steps related to certifications can cause difficulties in obtaining certifications or necessitate revalidation and reperforming process steps, wasting significant time, resources, and budget in the long term.
5. Increased turnover: In Company E, some individuals were frustrated by PD issues, particularly incompatibility between on-site and off-site teams, resulting in higher attrition rates. The lack of competence among off-site team members added to the frustration.

4.6.2 | Initial Waste

When process designers create a process, it can be the case that the process is suboptimal right from the beginning. This can be due to many challenges, for example, the difficulties of satisfying all the stakeholders involved. Although these effects can be multiple, we focus here on the effects that the initial waste has on the software developers.

1. Developers overhead: Besides tedious work and errors, several of the PD mentioned in the interviews directly caused an overhead for the software developers. Suboptimal documentation granularity, ID, the presence of unsuitable processes, and other activity-specific debt were the source of unnecessary effort spent by the developers in activities that were suboptimally designed.

TABLE 6 | Detailed issues leading to process debt (PD).

Issue	Description
Competences	An issue leading to PD is the lack of competences to design, manage, and execute processes. Processes can emerge and not be designed upfront, but regardless, it is important that processes are assessed, supported by infrastructure, and maintained. Another factor is the experience that practitioners already have with a given process, as well as with the domain and the product: if they are familiar with it, they might even optimize it without following all the instructions, while for employees that are not used to work according to a given process description, it might be confusing and may potentially trigger the introduction of PD.
Value neglection	An important issue about processes is that it is not often clear to all the stakeholders what value it brings to them or to the organization. The lack of a clear explanation and education of the stakeholders on what value the process brings to the organization often causes the process executors to neglect the process, leading to PD. On the other hand, a process needs to be designed with a clear purpose and value in mind and not just as a mandatory management equipment introduced by the organization.
Lack of follow-up assessment	Once the process has been designed and implemented, it should be assessed and improved according to the needs of the stakeholders. However, the lack of such follow-up does not allow us to identify and track possible PD. In addition, the lack of follow-up and improvement does not ensure that any divergence from the process is captured and handled, which may lead to the growth of PD.
Special contexts and situations	Different companies, units, teams, special domains, special events, etc., all contribute to making existing processes suboptimal, which generates PD and costly consequences if such special contexts and situations are not accounted for in the process.
Cultural causes	There are several cultural issues that have been mentioned by the practitioners. This category has been further elaborated in the following subcategories: A major issue is the lack of software culture in other organizations, for example, related to mechanical and electrical engineering. In addition, different teams and different individuals have different experiences, maturity, motivation, and leadership styles. This leads processes to be optimal for one team or one individual but not for another. Experienced people might neglect the process, thinking that they do not need it, while less experienced individuals and teams might not understand the purpose of the process, misjudging special situations. The developers might feel powerless to change processes: Although developers can be designers of their own processes in some cases, many times (especially for large organizations), they are actors or executors of processes that are designed by other stakeholders, sometimes in other organizations, for specific purposes (certification, safety, etc.). In these cases, developers are not often asked if the process is suited for them and their workflow, and they usually find it difficult to reach out and effectively ask to change the process, so PD is accumulated without being tackled effectively.
Lack of prioritization	Similarly to TD, PD issues are quite often not managed because they are not prioritized as important to be fixed. This can be due to too many foci for the teams or to strategies focused on short-term goals, etc. The interviewees mentioned that PD issues often get attention only after several employees have raised the issue, and one interviewee also stated that the PD issue needs time to be discussed iteratively during several occasions before it might get attention and thereafter be prioritized. In addition, many of the interviewees mentioned that PD issues are, at best, reported in a non-prioritized list also by the stakeholders, which does not help them get attention. This is an issue that we know well in connection with TD: to be prioritized, the TD issues need to be assessed and estimated. This is not something that is currently being done with PD issues, probably because of the lack of a framework, practices, and responsibility for implementing or changing the process.

(Continues)

TABLE 6 | (Continued)

Issue	Description
Technology and tools	The infrastructure is important to support the processes and to automate and facilitate their steps. The lack of infrastructure or the presence of infrastructure debt (described earlier) can cause additional PD, like in the case when an old tool configuration management tool does not support fast deliveries.
Cost of process changes	Sometimes, it is known how to eliminate or avoid some PD. However, the cost of removing the PD can be prohibitive, especially if the value and the interest paid by the practitioners due to PD are not clear and assessed. In addition, in many cases the estimations for changing the process turned out afterwards to be incorrect, which caused them to be cautious about fixing further PD.
Organizational causes	As for the cultural factor, there are many organizational issues that can affect the process. Examples of organizational issues are related to a structure where roles and responsibilities are not clear. This is directly responsible for roles' debt, as the roles created for the process do not have clear matches in the organization, and the process is not carried out by the individuals who are supposed to. Another issue is related to the interaction with external organizations with different cultures, interests, and power: Examples are processes (not) followed by open source organizations developing an OSS component used by the development team (e.g., the lack of a correct library versioning) or other stakeholders that may have interests in receiving data to compute analytics without knowing about the burden for the developers. The issue is the distance between the stakeholders in the software development team and external organizations. Several levels of indirection hinder making the stakeholders aware of each other's needs and challenges. Yet other organizational issues might be related to the sheer scale of the organization. The feedback loop between the process designers and the many stakeholders (e.g., many teams) is prohibitive, and efficient top-down follow-up is not possible. Tailoring the process to all possible contexts is not feasible without empowering the stakeholders themselves to tailor the process; however, this might easily diverge from the original purpose, as the stakeholders might try to optimize for their local optimum, creating PD that affects other stakeholders.
External trends	In some of the interviews, several of the participants described external trends that affect how processes are adopted in the company. Adopting processes that are not suitable for the company according to trends can lead to PD.
Lack of trust	Especially in the longitudinal study, we found that a lack of trust affected PD.

2. Developers' mistakes: Some PD affects the developers by making their workflow error-prone. For example, the frequent copy-pasting and duplication of information from one tool to another (because of ID) can cause mistakes that are then propagated to the stakeholders consuming the documentation created during the process. In general, when processes and workflow are not well synchronized (see synchronization debt), the developers often have to switch tasks continuously from one process to the other, one tool to another, etc. Although this does not perhaps directly create a measurable extra effort, developers suffer from unnecessary task switching as an additional layer of complexity on their job, which is not very different from having a more complex architecture to deal with. However, such additional complexity created by PD is difficult to see unless the workflow is visualized.
3. Developers' morale (tedious work): One issue, especially related to documentation debt and ID, is the creation of documentation or additional steps in the developers' workflow that need to be delivered but are unnecessary. This can happen for several reasons, as mentioned earlier, because of organizational or cultural issues or because of suboptimal infrastructure. These issues are the source of overhead and tedious work for software developers, which affects their morale. According to our interviews, it is commonly the case that developers prefer to work with processes that allow them to feel productive. The sense of accomplishment can be highly reduced by the feeling that much of the work has been dedicated to producing documentation that is not really useful for anyone. This is also related to the neglect of value and unclear purpose. In some cases, developers might not be aware of what the value of some of their processes is, which has the same effect on their morale, although the additional steps and documentation might not actually be unnecessary.
4. Uncertainty about the process: The lack of documentation and clarity about purposes and roles often creates confusion and uncertainty for the process executors. This leads to mistakes related to the outcome of the process, which in turn can generate several of the other consequences listed here, including the generation of additional PD. For

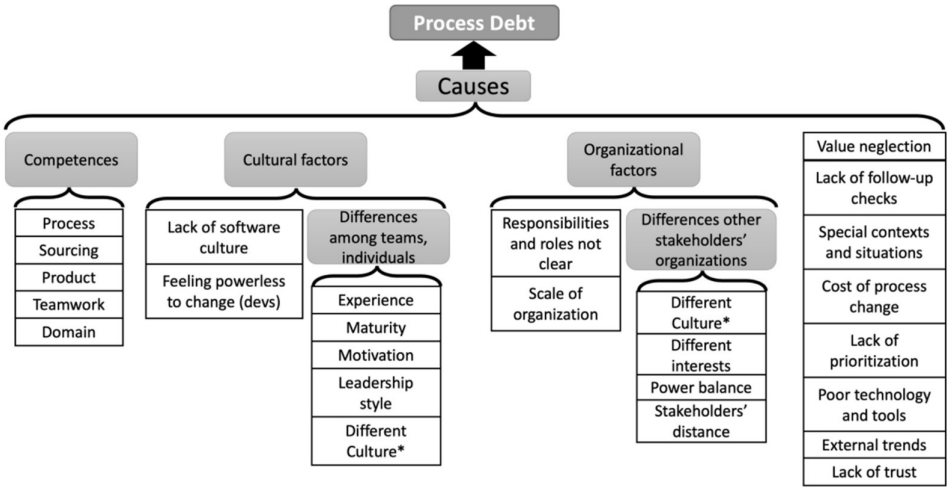

FIGURE 4 | A map of the causes of PD. Dark grey boxes have subcategories. "Different culture" is related to both teams and external stakeholders and therefore is marked with "*."

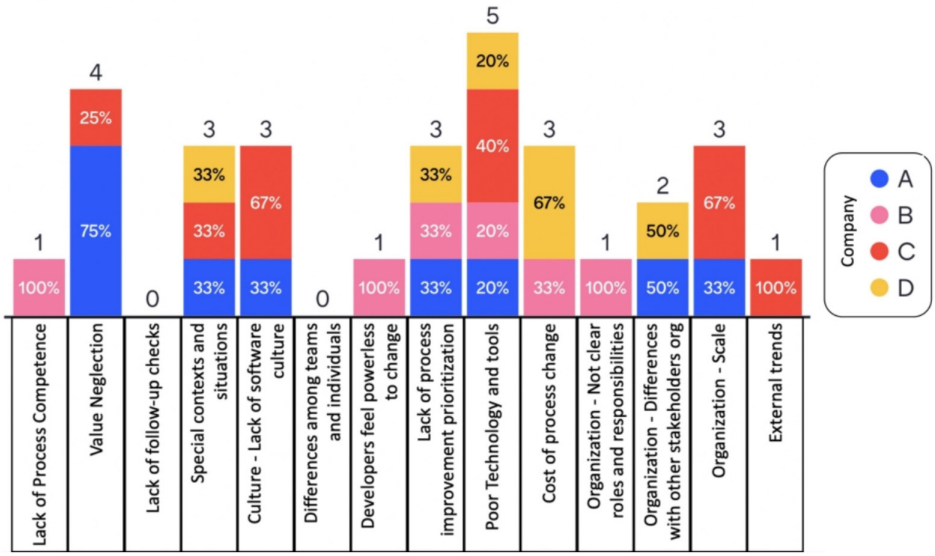

FIGURE 5 | The causes that were voted as the ones causing the most PD in the organizations. Respondents had to pick the three most important.

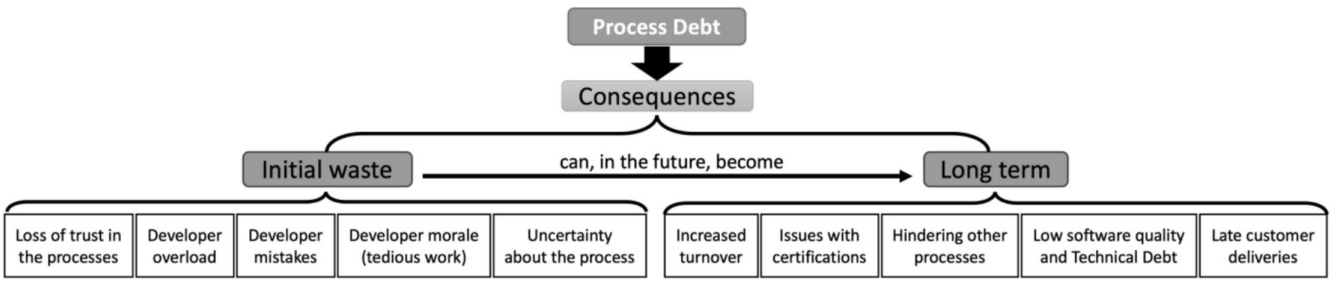

FIGURE 6 | Consequences of PD.

example, in Company E, the product owners (POs) we interviewed described that in one of the meetings, when they shared obstacles, they felt that managers always overreacted and immediately tried to solve the issues. Consequently, people stopped sharing concerns that they had, which reduced information flow and knowledge sharing. While the POs understood the meeting as a place to share and discuss problems—not necessarily fix them—the managers felt the pressure to solve all issues that could hinder the teams.

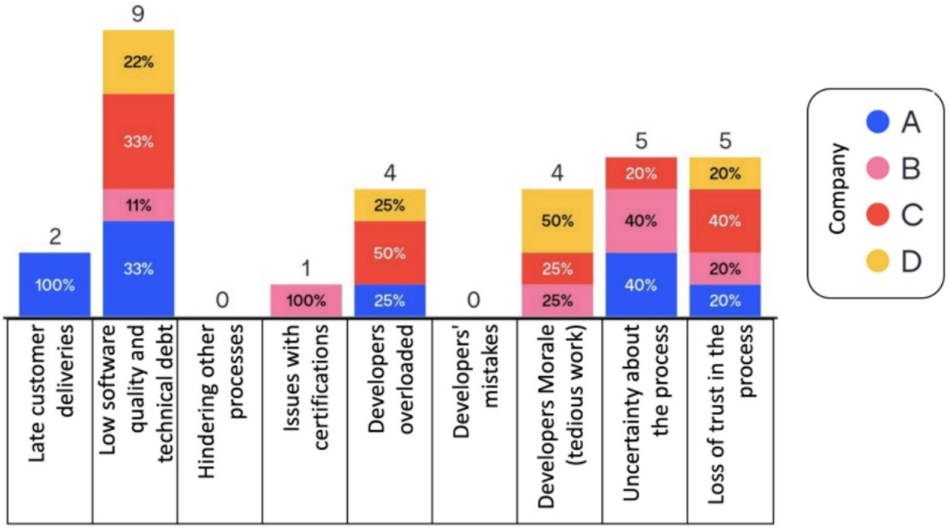

FIGURE 7 | The consequences of PD that were voted as the most harmful by the participants.

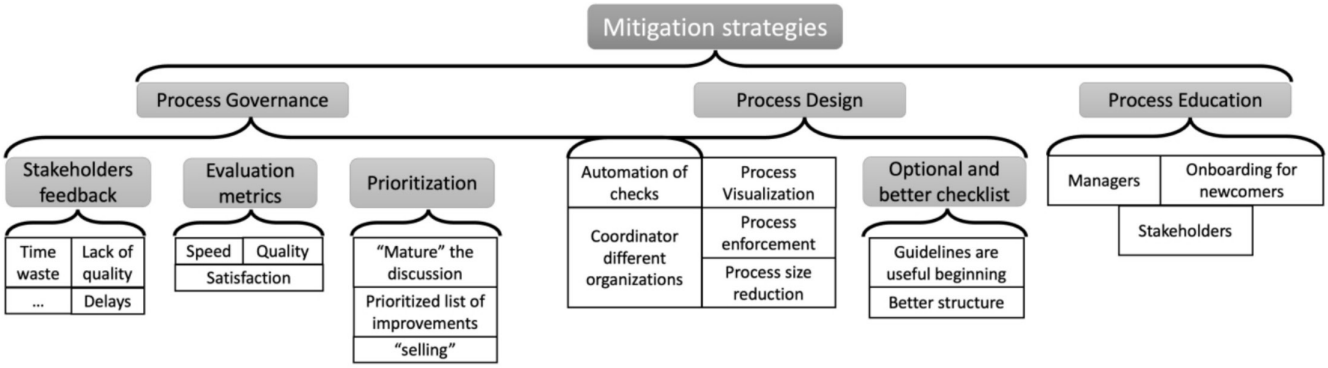

FIGURE 8 | Mitigation strategies proposed by the interviewees. Grey boxes have subcategories. "Automation of checks" and "coordinator different organizations" belong to two different categories.

5. Loss of trust in the processes: The presence of PD, especially related to the other initial waste categories mentioned previously, can create the perception for the developers that the processes are generally not trustable. Consequently, this can create a subculture in the teams for which processes are not worth being followed (even in those cases where the process might be optimal and valuable).

4.7 | Which Consequences Are the Most Harmful? (RQ2.5)

From the cross-company interview, we collected data as in Figure 7. Three voters for Company A, two for Company B, three for Company C, and two for Company D. The most harmful PD consequence was considered the one generating low software quality and TD. four or five participants (and generally voted by more than two companies) voted for uncertainty about the process, loss of trust in the processes, and impact on developers (morale and overload).

4.8 | How Can PD Be Managed? (RQ3)

We first answer the first research question, RQ3.1, about what mitigation strategies have been revealed by the practitioners during the interviews in Steps 1 and 2 (Figure 8), and then we report, according to our questionnaire in Step 3, which strategies are the most currently in use and which ones are still missing but organizations are moving towards implementing (RQ3.2).

We have previously reported the causes of PD. By mitigating such causes, organizations can also reduce the generation of PD. On the other hand, some of the mitigation strategies can be less proactive but more reactive.

From the interviews conducted in Step 1, we distilled three main categories of mitigation strategies: process governance, process design, and process education. These categories were validated in Step 2. In two cases, process governance and process design strategies overlap.

4.8.1 | Process Governance

These are strategies that can be employed by the process owners and stakeholders to reduce the chances of accumulating PD or to increase the chances of changing the processes to reduce PD.

4.8.1.1 | Stakeholders Feedback. One of the effective strategies to understand where PD hides in the organization is to ask the stakeholders and actors involved to report on hindrances such as delays, lack of quality, time waste, etc.

Some PD might be difficult to measure in practice, and the stakeholders are the only ones who can observe if there is an issue related to the process. Process owners should take the initiative to reach out to stakeholders, not only to set up the process but also to receive feedback from them during the continuous execution.

At the same time, it is also important that stakeholders are incentivized to report issues with the process, especially in explaining the impact of the issue.

4.8.1.2 | Evaluation Metrics. Measurements can be analyzed by the process owners, for example, understanding the speed with which the process is executed, the satisfaction of the stakeholders (for example, via surveys or direct contact), and the quality of the software that is implemented. Practitioners enumerate some solutions but also mention that current approaches are rather inefficient and better approaches need to be developed.

4.8.1.3 | Prioritization of PD Issues. Practitioners mentioned several ways in which they promote the allocation of resources to change processes and remove or mitigate PD.

First, like for TD management, they advocate the need for a prioritized list of PD issues: A generic and long list without having already provided a priority (e.g., based on cost/benefits analysis) by the makers of the list (process owners, developers, etc.) would not be taken into consideration by the responsible for allocating resources (PM, POs, etc.). It is more valuable to advocate for changing a few, well-supported PD issues rather than just having a long list of issues for which the RoI is uncertain and the time is not enough to discuss in meetings. However, even when the list is present, practitioners mention that the best strategy is to "sell" and "mature the discussion" around the PD issues that are the most critical to be mitigated during meetings with those responsible for allocating resources (PM, POs, etc.). Often, they mentioned, it takes several meetings before the participants of the meeting perceive a (PD) issue as important to be taken into consideration. Prioritizing PD is therefore perceived as mostly a social and "political" activity to lobby for the most important PD issues to be considered in a highly competitive race for resources (against features, TD, bug fixing, social issues, etc.).

4.8.1.4 | Process Enforcement. Some of the practitioners mentioned how enforcing the process with checks and tools can help reduce and avoid PD. Examples are obligatory forms to be filled in during the process to comply with an external certification process.

4.8.1.5 | Coordinator for Different Organizations. When different organizations require a process to be in place for the software teams (e.g., for safety certification and analytics), it is important to have an effective coordinator to bridge the needs of the two sides and to foster their communication.

This strategy overlaps with process design, as the coordinator can be a role that is included in the role definition of the process.

4.8.1.6 | Automation and Checks. The process owners and executors can decide to implement automated steps of the process and to build automated checks that the process is indeed followed.

However, such practice is not widespread: In many cases, automation requires an investment by a developer or a dedicated "process and tools" team to implement such automations. One of the issues mentioned by the interviewees is that such automations also need to be prioritized by the management to allocate resources, and in many cases, it is difficult to advocate for the benefits against the cost of implementation (or else, it is difficult to show that the principal paid would cover saving the interest).

Automation and checks are both governance strategies but can be part of the process design overall category, as their definition can be included in the design of a process.

4.8.2 | Process Design

These strategies are the ones that can be made when a process is either designed from scratch or when an emerging process is improved and optimized.

4.8.2.1 | Process Visualization. One of the ways to identify PD and argue for its reduction is to visualize the process. According to the participants, presenting the visualization to the managers can convince them that PD is detrimental and that it needs to be tackled. Process visualization can also include data-driven approaches. However, this does not seem to be currently an approach that is in use at the interviewed organizations.

4.8.2.2 | Process Size Reduction. Large organizations including several disciplines are characterized by the need to have formal processes that, in some cases, are predefined and hence the process steps are standardized: for example, a process to ensure that unit tests reach a given coverage might be more suitable for some teams but not others. A process, once standardized for the many software teams, might need to be reduced for some of the teams. In other cases, (emerging) processes tend to grow unchecked because of the needs of different organizations but need to be simplified.

4.8.2.3 | Optional and Better Checklists. To help executors follow the processes, there are often checklists with

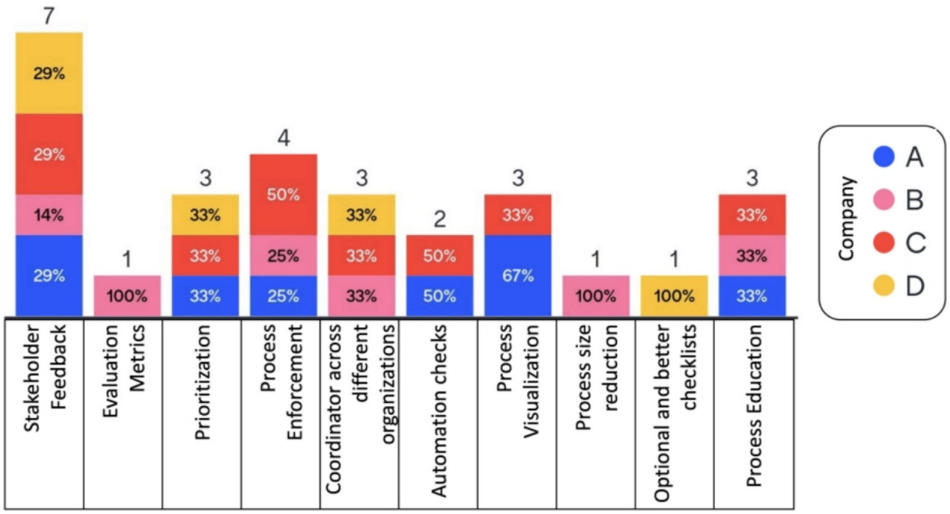

FIGURE 9 | Mitigation strategies mostly used in current practices.

the main steps of the process. However, practitioners mentioned that often such checklists are obsolete or not optimal and would need to be continuously updated.

Interviewees mentioned that checklists could be improved mainly in two ways: by restructuring overgrown checklists and by reducing them. In fact, checklists are especially useful the first time a process is in use, while after a while, experienced practitioners might just disregard the whole checklist as they feel like they have already learned the process, and they find following the whole checklist tedious and time-consuming. However, this can cause them to miss a critical step. By reducing checklists according to the experience of the team with the given process, we can therefore reduce the overhead of the teams and still allow practitioners to be reminded of critical steps that should not be overlooked.

4.8.2.4 | Process Complexity Reduction. In Company E, they decided to backsource the development to reduce the complexity caused by two incompatible development processes at two locations. By ending the outsourcing relationship, they could instead have a well-defined process in one place.

4.8.3 | Process Education

The last set of strategies to reduce PD is related to the education of processes. Executors, especially in large companies, need to be informed about not only how to follow a process but also about why processes need to be followed (value). In many cases, processes can be useful for one purpose and for a selection of stakeholders, but they are carried out by others. If these latter practitioners (e.g., developers in a team) are not informed and do not understand the value of such a process for the organization, they are not well motivated to follow the process.

4.8.4 | What Mitigation Strategies Do Companies Use Today, and Which Ones Are Still Missing?

To better understand which of the mitigation strategies mentioned by the interviewees are the most in use and which ones are not but should be more used, we have asked the participants of the cross-company interview in Step 3 to choose the three most used mitigation strategies today and the three ones that they would like to implement in the future. Figures 9 and 10 show what the respondents have answered.

The most striking results from looking at the results are that currently, the organizations are heavily using stakeholder feedback (seven answers), followed by process enforcement, prioritization techniques, coordination across different organizations, process visualization, and process education (all with three answers each).

On the other hand, we can see how the use of automation and checks grows from two answers (used today) to seven answers (wished to be applied in the future), while process visualization is increased from 3 to 5 answers, process size reduction increased from 1 to 4, and evaluation metrics from 1 to 3. These are the strategies that are not yet fully in place, but organizations are willing to bet on for the future. These also constitute ideal research goals for future research on this topic.

4.9 | Time Dimension (RQ4)

To illustrate how PD emerged and changed over time, we will describe how continuous processes were introduced in Company E.

4.9.1 | Introducing Continuous Processes Reveals PD

Disconnected and incompatible processes: Processes like DevOps and BizDev were introduced to address disconnected activities and become more customer-driven. An internal document stated, “Change is needed to better serve our customers. We need to change the way we work to ensure that we can serve our customers with what they want in a timely manner.”Traditionally, separated processes were replaced with continuous ones that linked planning, testing, and development. This shift introduced synchronization debt as these

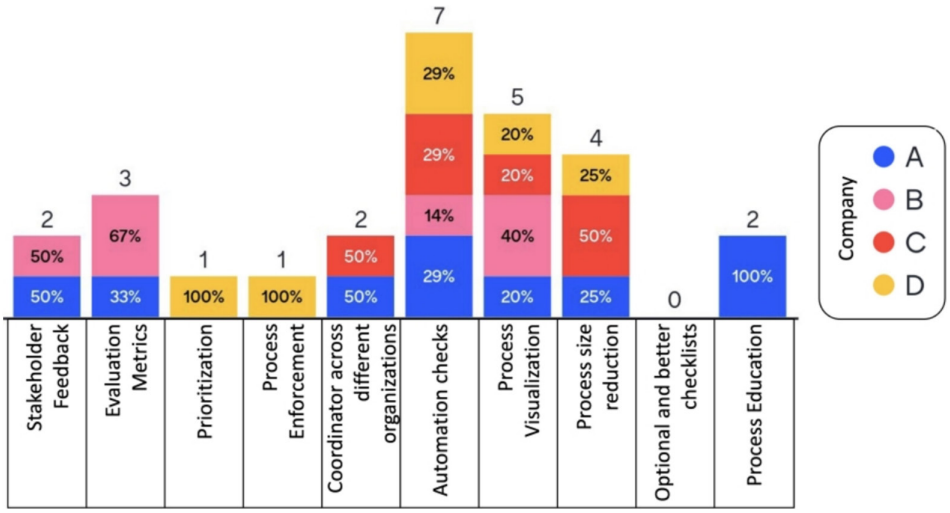

FIGURE 10 | Mitigation strategies that are wished to be applied in the future by the participants.

processes, initially optimized independently, now required integration.

At the start of the study in Company E, business development and software development operated separately, with business developers setting priorities for software developers. Consequently, each side optimized its processes, creating synchronization debt. The business side initiated projects that could not be developed from a technical standpoint, while developers often lacked timely feedback on necessary changes because the business developers were hard to reach. This caused the time from identifying a feature to delivering the feature to take much longer than necessary.

4.9.1.1 | Organizational Changes. To mitigate the synchronization debt, the company decided to create agile teams with team members from both business and development (BizDev teams), including testers and user experience designers, to achieve a continuous planning and execution process.

4.9.1.2 | Process Unsuitability. However, the new working process and the different working cultures between the two groups introduced new process unsuitability. First, it quickly became evident that different roles had different needs in how they performed their tasks. For example, business developers appreciated a very open work area that allowed discussions and quick feedback. Software developers, in contrast, wanted to have designated seats because their desktop computers contained specific hardware and multiple monitors. In addition, they needed a quiet working zone and wanted to protect their time to focus on technical programming tasks. Even though the BizDev setup increased the speed of feedback and clarifications, frequent interruptions fragmented the developers' day, and the developers perceived that their personal productivity was reduced. These differences in cultural factors, interests, and a lack of follow-up checks caused PD. Further, due to the large team sizes (13–20 members) and diverse roles, some meetings, such as standups, lacked relevance for all participants, leading to decreased motivation and perceptions of wasted time.

4.9.2 | Sourcing, Competence, and Unsuitable, Continuous Processes Reveal PD

4.9.2.1 | Introducing Sourcing. The lack of resources in Norway made it challenging to scale. Therefore, to develop new features fast enough and not lose market opportunities, Company E decided to outsource, and the company made an agreement with an offshore outsourcing company in India. The Norwegian teams used Kanban, while the Indian teams followed Scrum. The outsourcing caused several types of debts, such as competence debt, synchronization debt, and unsuitable processes. For example, because of legal requirements in the final delivery, the onshore teams were required to do code reviews on all code from the outsourcer. Because they followed different processes, it was discussed if the code review practice should be continuous or timeboxed.

4.9.2.2 | Suboptimal Processes. The company decided that an optimal process would be to make the code reviews timeboxed. Reasons included being able to schedule enough time for doing QA and reducing the number of interruptions for the Norwegian developers. This process turned out to be suboptimal. Every second week, because of weak domain competence at the offshore team, the Norwegian developers received large pieces of code. Because the offshore team worked for 2–3 weeks before the PRs were sent, the code sent for review was large and complex. Consequently, reviewing the large PRs was time-consuming, tedious, and caused frustration.

It became evident that the quality work took a considerable amount of time from the Norwegian developers, reducing the productivity of the onsite developers so much that the total productivity became lower than before adding the extra offshore developers. A developer explained: "Sometimes it takes such a long time that I actually have to write the correct code, send it over to them, and ask them to implement it."

4.9.2.3 | Unsuitable Processes. Even though the problems of the pull request process were well known by the onsite

developers, for the managers, it all looked fine, and consequently, no measures were taken. One reason it looked fine was the key performance indicator (KPI) measurements. While evaluation metrics can be a good process governance approach to mitigate PD, they can also be misleading, creating PD of type "unsuitable process." For example, the offshore teams were measured by KPIs set by the company, and these were related to numbers they received from queries in Jira (e.g., how long a task had stayed at a certain stage in the development process). So even though many pull requests were rejected and the quality was not seen as satisfactory, the KPIs on the code and productivity were met, resulting in happy managers but frustrated onsite developers.

> The offshore teams are measured by the number of pull requests that are approved or not, but for some reason that measurement is calculated by completed Jira tasks and not actual PRs in the system where we approve the code. Often, I feel that one pull request has ten attached Jira issues, to double the KPI. I think measuring an outsourcing partnership based on the number of approved PRs is not a good solution.

4.9.2.4 | Redesigning Processes. Consequently, there was a delay in addressing and solving the well-known problem. Eventually, to mitigate the challenges of the offshoring set-up, the company introduced additional quality meetings where they looked at the PRs and discussed them. Further, they introduced Slack as a tool to make communication more informal and faster and, therefore, more quickly, could clarify issues related to PRs. The situation improved a bit.

4.9.3 | New Roles and Roles Debt

In the longitudinal study, all types of PDs were observed, primarily identified through retrospectives in the company and better understood in the interviews.

4.9.3.1 | Roles Debt. Synchronization and alignment debts were apparent early in the study and persisted for years. In Company E, the PO role was introduced with insufficient explanation, leading to confusion. It was believed that a short memo was enough. The PO's responsibilities were mainly related to being a team leader. Many people assigned the PO role misunderstood their responsibilities as they were familiar with the PO role in Scrum. Further, as there were many POs, they needed to collaborate. However, because they had a different understanding of the role, communication and collaboration problems emerged and lasted for months. In a retrospective with the managers and the POs, the role ambiguity was identified, and a detailed role description was created, emphasizing team leadership, business prioritization, and stakeholder coordination. However, the extensive coordination required led to the introduction of a coordinator role to represent POs in management meetings. This change inadvertently slowed PO work due to their exclusion from critical decision-making, leading to misunderstandings.

4.9.3.2 | Redesigning Processes. Again, the process and roles needed to change. After a few months, the coordinator role was removed. The POs were again included in the weekly prioritization and sync meetings with the management, and a new process was designed for this meeting. As POs gained experience over a year, the process evolved to grant them more decision-making authority.

5 | Discussion

5.1 | Contributions and Implications

PD is a phenomenon that has not been studied as much as TD. Through a 3-step investigation, we have collected qualitative data from five companies and quantitative data from four of them for validation purposes.

The main contributions are:

- a comprehensive framework to practically and theoretically reason about PD, including:
  - causes, consequences, and types of PD
  - occurrence patterns and evolution of PD over time
- a survey of the state of practice for PD management in five companies
  - evidence of concrete instances covering each type of PD
  - mitigation strategies to manage PD
- extensive validation of results by triangulating methods and sources across contexts

We discuss these points in the following sections.

5.1.1 | A Framework to Reason About PD

We propose an overall PD conceptual framework (Figure 2) where the key elements and their relationships are visible. We also give detailed taxonomies of types of PD, occurrence patterns, causes, effects, and mitigation strategies. Our framework can support reasoning about PD for practitioners who would like to manage PD systematically and need a comprehensive reference point. First and foremost, TD research has shown that the first important step to managing debt is to build awareness around it and to start a discussion in the organizations.

From the theoretical point of view, our framework is not complete, as the taxonomies can be extended with additional evidence collected from new and specific case studies. However, our results show that, when extending our initial framework with new, in-depth evidence from a longitudinal case study with a different context from the initial four companies, the framework was substantially validated with some little additions. This gives confidence that the framework represents a robust first step towards building a comprehensive theory of PD and is already usable as a key theoretical reference point.

Our findings show that the PD lens can enrich the knowledge related to SPI with a new perspective on how to prioritize SPI

work. Much of the existing literature is dedicated to assessing SPI or to improving specific processes; however, little is found about how to prioritize if an SPI intervention is actually more important than another one or even if improving a process is more important than developing features and fixing bugs. The debt perspective and our characterization using entities such as principal and interest, borrowed from the financial domain, can help to reason about the prioritization of SPI to avoid or repay an existing PD that is particularly disruptive, while avoiding SPI work and spending resources that would reduce PD with limited negative effects.

5.1.2 | Defining PD

Our investigation shows that PD is an existing phenomenon that can cause a huge amount of negative effects, including the limitation of applying modern practices to support vital business approaches such as continuous delivery. PD is also recognized as generating TD.

Although a quantification of the (negative) effects of PD is not provided in this study, according to the interviewees, the magnitude of its impact is comparable to the TD one. Further studies and surveys should be conducted on the PD phenomenon to assess its actual impact. Our taxonomies can help create a solid instrument to investigate such a phenomenon.

Throughout our report, we have mentioned several concrete instances shared by the practitioners supporting our taxonomies. Besides underpinning our framework, such instances constitute a concrete list of examples of PD that can occur in organizations. Practitioners can also use our taxonomy and even such a concrete list of instances to identify and differentiate across PD types in their context, to recognize their causes and effects even before PD becomes disruptive, and to be able to reason about prioritizing PD's avoidance and removal.

A key point revealed by observing PD in the longitudinal study is that removing PD often introduces new PD. This is different from TD, where repaying TD often implies that the new solution is better (excluding a few corner cases). This means that evaluating if repaying PD is worth it requires a more complex assessment, and more than one "repayment" might be included in the equation to reach a substantial improvement. More research is needed to understand such evaluation and assessment. This is in line with Argyris and Schon's concept of double-loop learning [23], where additional evaluation is done after a solution is applied.

In this paper, we created a first framework to reason about PD, and we have therefore looked into the commonalities across several companies. However, the different contexts should be further studied to understand what makes PD and its management differ across companies.

5.1.3 | PD Mitigation Practices

The reported mitigation strategies used by the practitioners to manage PD can also be applied in other contexts (if not precisely, at least they can give inspiration for new practices). Additionally, besides what is used today, we show what mitigation strategies are in practitioners' plans for better managing PD in the future. This can give key targets for companies developing tools for SPI and researchers to investigate novel approaches along the directions mentioned by our informants.

As for today's state of practice, it seems that the interviewed companies rely substantially on stakeholder feedback as the main mitigation strategy, followed by process enforcement, extensive coordination across different organizations, and education. However, for the future of mitigating PD, companies envision an increase in automated checks, process visualization, process reduction, and, to a smaller extent, evaluation metrics. Conversely, they envision a reduction in stakeholder feedback, prioritization, process enforcement, education, and coordination. In summary, companies wish to move from manual, tedious, and error-prone management of PD to a more lightweight, automated, and visual approach, which entails the need for an infrastructure to support such approaches. This also provides a roadmap for future research: automating and visualizing PD and creating methods and tools to support process stakeholders is the priority for impactful research in this field. A clear starting point is to collect and visualize existing methods in dashboards and iterative retrospectives to reason about the existence, causes, and consequences of PD.

In this paper, we have provided examples and trends related to the overall categories of mitigation strategies. However, the occurrence of PD and mitigation practices need to be better contextualized and aligned with the lifecycle of specific processes or debt management in a company. Future work, especially including case studies where PD is analyzed and resolved in the context of a specific process and organization, can shed more light on how to make the mitigation practices more applicable and actionable. Important variables to study alongside PD occurrence include the composition of stakeholders involved in the process, the company's size, software release frequency, and budget for assessing, analyzing, and solving PD. These variables can significantly influence PD occurrence and mitigation, and studying different organizations may yield a different picture of the state of practice for PD and future priorities for research.

5.2 | Process vs. TD

The analogy of debt applies to PD much like it does for TD, considering the principal amount and especially interest as pivotal concepts. Interviewees found this metaphor helpful for better prioritizing improvements in processes. Our findings align with existing knowledge on TD.

However, we noticed several distinctions that justify further research on this subject. We recognized that a key aspect of PD is its value for stakeholders, a third factor that must be included in evaluating PD. This consideration could potentially be incorporated into TD research. Furthermore, poorly designed processes may generate initial wastage, which is not typically observed in TD.

Although there are areas where causes and consequences of PD and TD overlap, we discovered they largely differ. We also

observed trends for accumulation and subsequent triggers for PD: flawed processes developed by designers, deviations from well-planned processes by executors, and the shortcomings of supportive infrastructure might present challenges that go beyond areas affected by TD.

In terms of mitigation strategies, key principles include the necessity to prioritize PD in a backlog and to educate practitioners about PD and TD. However, various distinguishing factors emerged, such as the need for stakeholder feedback, cross-organizational coordination, and more.

This implies that PD should be studied as a distinct phenomenon from TD. We also identified various kinds of PD, defined by specific activities and suboptimal process aspects, which calls for a more detailed classification in future research. Existing strategies for TD could also be adapted to foster PD research.

5.3 | Validity, Reliability, and Limitations

Our study relies on interviews, observations, and quantitative data collection with a limited set of organizations. This means that our results cannot be considered general and suffer from external validity threats [30]. For example, additional organizations with different domains and contexts should be interviewed and surveyed to reach a complete conceptual framework. Some of our results would hold in different settings (e.g., the types of PD), but some of the causes and consequences might not apply or differ in such contexts. We, therefore, acknowledge that further research is required. However, we argue that our exploratory effort is already valuable, as we included different companies with diverse contexts and stakeholders, ranging from process designers to software developers, team leaders, and practitioners in other organizations, such as hardware designers.

To mitigate the threats to external validity, we have also applied two additional validation steps: running a workshop, collecting additional qualitative and quantitative data, and comparing our results to the in-depth findings from a longitudinal case study in a new and different Company E. We argue that it would not be possible to cover the phenomenon in its entirety with just one study, and we call for the software engineering community to contribute to studying PD with additional investigations.

As for reliability, although qualitative studies often rely heavily on the researchers' interpretation, we have used several mitigation strategies to limit the bias introduced by the authors. First, in the majority of cases, at least two researchers were present at the interviews, and two researchers analyzed the interviews in parallel, discussing and agreeing on the resulting coding. At least two researchers have discussed the findings and definitions across studies to ensure that data from the longitudinal study would be correctly interpreted and mapped to the framework obtained from the first phase.

One limitation is the lack of input for the workshop in Phase 3 from practitioners in the longitudinal case study related to Phase 2, focusing only on the first four companies investigated in the first phase. However, similar insights were collected and analyzed in the longitudinal case study with a different method; the only difference is the data collection format.

6 | Conclusion

In this paper, we extensively explored the phenomenon of PD in five companies through three steps, including interviews, workshops, direct observations over a longitudinal case, and cross-case validation comparisons.

The current state of the art and practice of PD lacks a solid framework to reason about the concept and empirical evidence of the state of the art and practice. Consequently, the phenomenon is overlooked, and software organizations do not have a reference point to manage PD.

This paper presents a novel framework to categorize PD and its empirically based state-of-the-art and practice concepts, providing a clear reference point for practice and future research.

The framework also defines the interrelations between various elements of PD and emphasizes that managing PD is a cyclical, iterative process. The framework can be used to analyze and structure PD initiatives, suggest improvements, and provide guidance for decision-making.

We report a state of practice including concrete instances of PD, causes, consequences, occurrence patterns over time, and mitigation strategies to manage PD, which can help practitioners systematically manage PD in practice. The interviewees also give valuable insights on what is needed in the field to manage PD better, supporting further research and tool development on the topic.

We found that the debt metaphor helps reason about the prioritization of process improvements concerning competing activities such as feature development. PD is a whole new field with respect to TD, with little overlap. In this paper, we provide the foundations for the development of this phenomenon. We encourage and envision a larger and joint community effort to further develop theories related to PD and to improve its state of the art and practice.

Acknowledgments

We thank the interviewees from the Software Center companies (Companies A–D) for the invaluable insights provided, as well as the participants in Company E. This research was partly supported by the Research Council of Norway through Grant Numbers 321477 and 340991.

Conflicts of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper. The authors declare the following financial interests/personal relationships, which may be considered as potential competing interests: Jan Bosch reports financial support was provided

by Software Center (industrial consortium in Sweden). Nils Brede Moe and Viktoria Stray report financial support provided by the Research Council of Norway.

Data Availability Statement

Research data are not shared.

References

1. T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A Decade of Agile Methodologies: Towards Explaining Agile Software Development," *Journal of Systems and Software* 85 (2012): 1213–1221.

2. O. Pedreira, M. Piattini, M. R. Luaces, and N. R. Brisaboa, "A Systematic Review of Software Process Tailoring," *SIGSOFT Software Engineering Notes* 32 (2007): 1–6, https://doi.org/10.1145/1241572.1241584.

3. R. G. Schroeder, K. Linderman, C. Liedtke, and A. S. Choo, "Six Sigma: Definition and Underlying Theory," *Journal of Operations Management* 26 (2008): 536–554.

4. Institute, S.E., "CMMI for Development, Version 1.3," Technical Report. Carnegie Mellon University. Pittsburgh, Pennsylvania, 2010, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9661. Technical Report CMU/SEI-2010-TR-033.

5. S. Zopf, "Success Factors for Globally Distributed Projects," *Software Process: Improvement and Practice* 14 (2009): 355–359.

6. D. Moitra, "Managing Change for Software Process Improvement Initiatives: A Practical Experience-Based Approach," *Software Process: Improvement and Practice* 4 (1998): 199–207.

7. T. Besker, A. Martini, and J. Bosch, "Software Developer Productivity Loss due to Technical Debt—A Replication and Extension Study Examining Developers' Development Work," *Journal of Systems and Software* 156 (2019): 41–61, https://doi.org/10.1016/j.jss.2019.06.004.

8. T. Besker, H. Ghanbari, A. Martini, and J. Bosch, "The Influence of Technical Debt on Software Developer Morale," *Journal of Systems and Software* 167 (2020): 110586, https://doi.org/10.1016/j.jss.2020.110586.

9. Z. Li, P. Avgeriou, and P. Liang, "A Systematic Mapping Study on Technical Debt and Its Management," *Journal of Systems and Software* 101 (2015): 193–220, https://doi.org/10.1016/j.jss.2014.12.027.

10. A. Martini, V. Stray, and N. B. Moe, "Technical-, Social- and Process Debt in Large-Scale Agile: An Exploratory Case-Study," in *Agile Processes in Software Engineering and Extreme Programming—Workshops* (Springer International Publishing, 2019): 112–119, https://doi.org/10.1007/978-3-030-30126-2_14.

11. P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)," in *Dagstuhl Reports* (Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016): 110–138.

12. D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "What Is Social Debt in Software Engineering?" in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)* (IEEE, 2013): 93–96, https://doi.org/10.1109/CHASE.2013.6614739.

13. K. H. Rolland, L. Mathiassen, and A. Rai, "Managing Digital Platforms in User Organizations: The Interactions Between Digital Options and Digital Debt," *Information Systems Research* 29 (2018): 419–443, https://doi.org/10.1287/isre.2018.0788.

14. A. Martini and J. Bosch, "An Empirically Developed Method to Aid Decisions on Architectural Technical Debt Refactoring: AnaConDebt," in *Proceedings of the 2016 38th International Conference on Software Engineering* (IEEE, 2016): 31–40, https://doi.org/10.1145/2889160.2889224.

15. I. Sommerville, *Software Engineering* (Pearson Education, 2007).

16. S. Kupper, D. Pfahl, K. Jurisoo, P. Diebold, J. Munch, and M. Kuhrmann, "How Has SPI Changed in Times of Agile Development? Results From a Multi-Method Study," *Journal of Software: Evolution and Process* 31 (2019): e2182, https://doi.org/10.1002/smr.2182.

17. P., Veyrat, "What are the differences between workflow and processes?," 2018, https://www.heflo.com/blog/bpm/workflow-and-processes/. hEFLO BPM Blog.

18. M. Unterkalmsteiner, T. Gorschek, A. K. M. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt, "Evaluation and Measurement of Software Process Improvement—A Systematic Literature Review," *IEEE Transactions on Software Engineering* 38 (2012): 398–424, https://doi.org/10.1109/TSE.2011.26.

19. M. Kuhrmann, P. Diebold, and J. Munch, "Software Process Improvement: A Systematic Mapping Study on the State of the Art," *PeerJ Computer Science* 2 (2016): e62.

20. P. Clarke and R. V. O'Connor, "The influence of SPI on Business Success in Software SMES: An Empirical Study," *Journal of Systems and Software* 85 (2012): 2356–2367.

21. N. B. Moe, "Key Challenges of Improving Agile Teamwork," in *Agile Processes in Software Engineering and Extreme Programming: 14th International Conference, XP 2013, Vienna, Austria, June 3–7, 2013. Proceedings 14* (Springer, 2013): 76–90.

22. R. van Solingen, E. Berghout, R. Kusters, and J. Trienekens, "From Process Improvement to People Improvement: Enabling Learning in Software Development," *Information and Software Technology* 42 (2000): 965–971.

23. C. Argyris, *Organizational Learning II. Theory, Method, and Practice* (Addison-Wesley, 1996).

24. A. Martini, T. Besker, and J. Bosch, "Process Debt: A First Exploration," in *2020 27th Asia-Pacific Software Engineering Conference (APSEC)* (IEEE, 2020).

25. M. Mikalsen, N. B. Moe, V. Stray, and H. Nyrud, "Agile Digital Transformation: A Case Study of Interdependencies," in *Proceedings of the 39th International Conference on Information Systems (ICIS)* (Association for Information Systems (AIS), 2018).

26. V. Stray, N. B. Moe, M. Mikalsen, and E. Hagen, "An Empirical Investigation of Pull Requests in Partially Distributed BizDevOps Teams," in *2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE)* (IEEE, 2021).

27. R. K. Yin, *Case Study Research: Design and Methods*, vol. 5 (Sage, 2009).

28. G. Walsham, "Interpretive Case Studies in IS Research: Nature and Method," *European Journal of Information Systems* 4 (1995): 74–81.

29. H. K. Klein and M. D. Myers, "A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems," *MIS Quarterly* 23 (1999): 67–93.

30. P. Runeson and M. Host, "Guidelines for Conducting and Reporting Case Study Research in Software Engineering," *Empirical Software Engineering* 14 (2009): 131–164.