Process Debt: Definition, Risks and Management

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

[1]University of Oslo
[2]RISE Research Institutes of Sweden AB
[3]SINTEF
[4]Chalmers tekniska hogskola - Campus Lindholmen

April 08, 2024

Abstract

Context: Process Debt, like Technical Debt, can be a source of short-term benefits but often is harmful in the long term for a software organization. Nonetheless, information about Process Debt is scarce in the current literature. *Objective:* This paper aims to define Process Debt, describe its occurrence with associated risks, and show examples of mitigation strategies. *Method*: Firstly, we conducted an exploratory study of Process Debt in four international organizations by interviewing 16 practitioners. Then, we validated and extended the findings with a cross-company focus group with additional 10 practitioners. Finally, we analyzed 58 additional observations and 35 interviews from a longitudinal case study. *Results*: The findings show that Process Debt can be a harmful phenomenon that needs attention. We provide a framework and a definition, and report causes, consequences and occurrence patterns over time of Process Debt. We present mitigation strategies and which ones need further attention for future research. *Conclusions*: 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.

Process Debt: Definition, Risks and Management

Antonio Martini

University of Oslo antonima@ifi.uio.no

Viktoria Stray

University of Oslo and SINTEF

Terese Besker

RISE Research Institutes of Sweden

Nils Brede Moe

SINTEF

Jan Bosch

Chalmers University of Technology

Abstract

Context : Process Debt, like Technical Debt, can be a source of short-term benefits but often is harmful in the long term for a software organization. Nonetheless, information about Process Debt is scarce in the current literature.

Objective: This paper aims to define Process Debt, describe its occurrence with associated risks, and show examples of mitigation strategies.

Method : Firstly, we conducted an exploratory study of Process Debt in four international organizations by interviewing 16 practitioners. Then, we validated and extended the findings with a cross-company focus group with additional 10 practitioners. Finally, we analyzed 58 additional observations and 35 interviews from a longitudinal case study.

Results : The findings show that Process Debt can be a harmful phenomenon that needs attention. We provide a framework and a definition, and report causes, consequences and occurrence patterns over time of Process Debt. We present mitigation strategies and which ones need further attention for future research.

Conclusions : 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.

Keywords: Software Development, Process Debt, Software Process Improvement, Qualitative Study

1. INTRODUCTION

Software Process Improvement is a widely studied area in software engineering. Finding systematic and efficient ways to produce software that effectively delivers business value to their 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 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 continuously to design and tailor new and efficient processes 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. 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 sub-optimal 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, e.g. how to best prioritize the improvements of the 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 sub-optimal 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 25% 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 sub-optimal processes that needed to be fixed. In other words, they reported instances of PD that had a direct negative consequence to the developers

and their work. Such 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 Process Debt?

RQ1.1: How can Process Debt be defined?

RQ1.2: What types of Process Debt exist?

RQ2: How does Process Debt occur?

RQ2.1: How is Process Debt accumulated?

RQ2.2: What are the causes of Process Debt?

RQ2.3: What causes are the most common?

RQ2.4: What are the negative consequences of Process Debt?

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? (results from interviews, longitudinal and workshop)

RQ3.2 What mitigation strategies are still missing to manage PD?

RQ4 Time dimension (results from longitudinal case study)

R4: How did PD change over time, and how was it managed?

First and foremost, PD needs to be understood and defined. This would help facilitate reasoning and provide a clear reference point of PD (RQ1). Then, we need to understand how PD occurs in practice, including causes and consequences, and which types of PD are more frequent and impactful. In other words, we need to understand the state of practice of PD (RQ2), which would help both recognize PD for practitioners as well as present a baseline to collect novel results in the future and develop novel techniques. In this regard, PD needs to be managed, which requires mitigation strategies to avoid, repay or mitigate the occurrence of PD (RQ3): we need to understand both the state of the practice, but also what is needed in the foreseeable future. Finally, we need to understand how PD occurs and is explicitly managed by looking at its evolution over time (RQ4).

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 software process improvement. 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. BACKGROUND

2.1 Process Debt and Technical Debt

Technical Debt (TD) research has increased steadily in the last ten years, showing great interest both from academia as well as the industry [9]. However, the TD metaphor is mainly restricted to enclose only technical issues, such as those related to code, tests, architecture, documentation, etc. [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 [10]: PD is "a sub-optimal 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.

Same as for 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 sub-optimality (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 sub-optimal 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 sub-processes 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 [17].

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 [16]. 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, or 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 process debt

In software development, there is a long tradition of regularly improving the processes [26] Software Process Improvement (SPI) is about making the software processes better and is mainly a human activity [18]. SPI has been found to be positively associated with business success in small to medium-sized companies [19]. However, process problems are frequently identified but rarely solved [20], 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 [27] since 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 Schön [21] distinguish between two types of learning in the organization, which they call "single loop learning" and "double loop learning": a single loop learning is a change in practice in the event of a problem, in order to avoid that same problem for the future. Single-loop learning involves changing processes when a problem occurs in order to prevent the same problem in the future. Examples include managers monitoring development costs, sales, or customer satisfaction to ensure that organizational activities remain within predetermined limits to keep the organization "on track." In single-loop learning, actions are modified slightly to attain the desired results if their consequences are not met. There is a feedback loop between observed effects and modifications or refinements that influence these effects. On the other hand, double-loop learning is when time is taken to understand the factors that influence the effects, and the nature of this influence is called the governing values [21]. It involves using the problems being experienced to understand their root causes and then take actions to address these causes. One example is what happens when a software error is corrected. Correcting the error itself can be seen as single-loop learning, but if something is done with whatever caused the error to be introduced, that is considered 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 sub-optimal 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 need to "learn how to learn", also known as deutero-learning [21]. 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 software process improvement 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 software process improvement 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 based on an additional empirical extension of the previous study, where this manuscript includes an in-depth value-added analysis of the results derived from the first study to provide a more detailed and comprehensive understanding and perception of the concept of PD.

This manuscript has been extended to include six additional research questions (RQ2.3, RQ2.5, RQ3, RQ3.1, RQ3.2 and RQ4), where the results of these questions are derived from a totally new set of independent data collection.

This extended manuscript includes the results from a follow-up cross-company interview with company A-D, which reports a validation of the results from the previous study and additional insights coming from a close-ended questionnaire on the extent to which the causes, consequences and mitigation strategies are the most crucial for the interviewed companies (new RQ2.3, RQ2.5, RQ3, RQ3.1, RQ3.2). Further, the current study includes data collected during a six-year-long case study focusing primarily on software process improvements on an additional company (Company E), which was primarily used to validate and complement the results using a new method and data source as well as for adding a new RQ related to the time dimension of PD (RQ4).

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] (see section 2.4). 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 5-6). Figure 1 also describes what activities were performed in each of the phases, together with the research methods used in each phase.

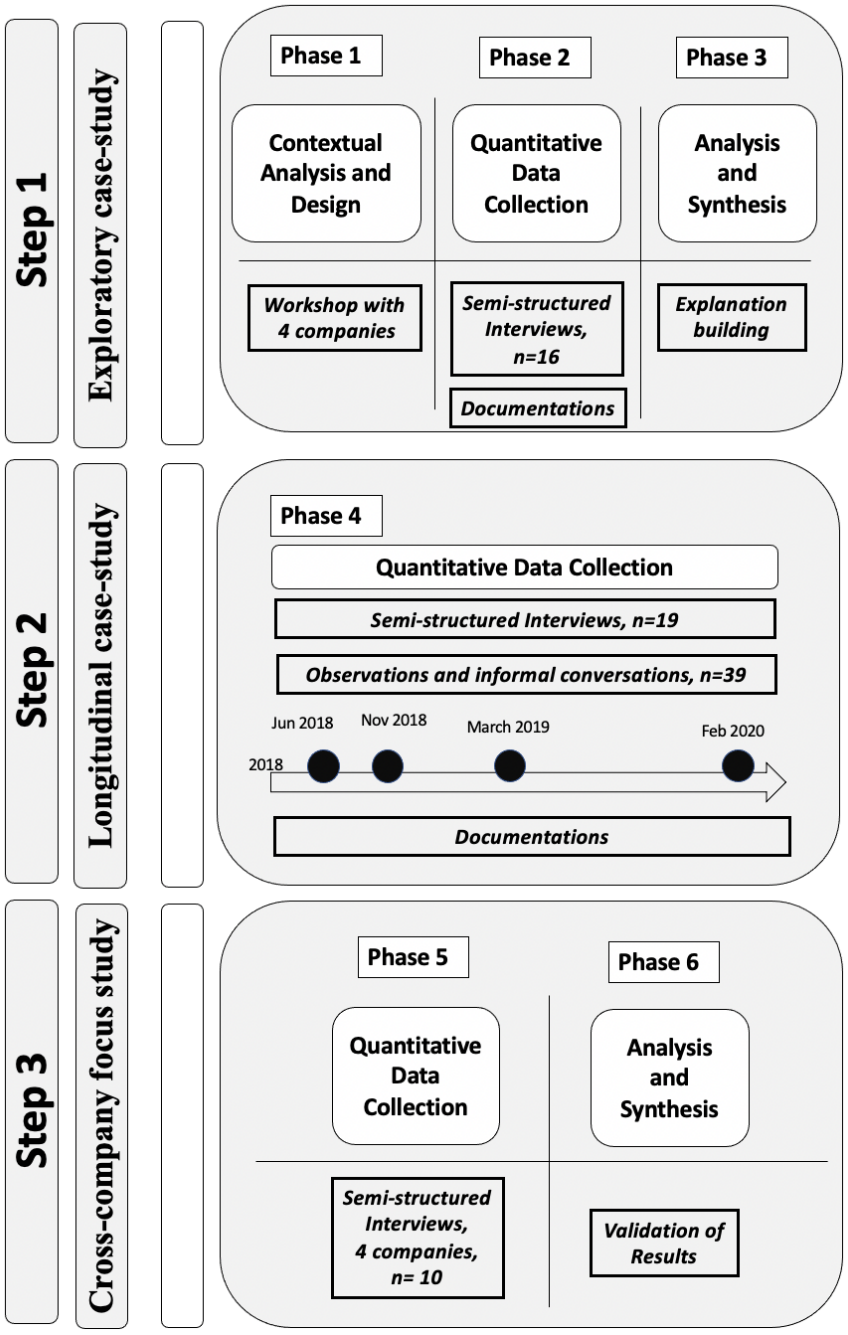

Figure 1. Visualization of the research model and method used in each phase.

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, therefore, employed a multiple qualitative case study approach [1] to gain a detailed understanding of how PD arises and its related adverse consequences.

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 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 minutes and was digitally recorded.

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 the technique of unstructured and semi-structured interviews where the questions were both formulated as general concerns and interests from the researcher about PD. However, the interviews also had several questions that were prepared and formulated in advance [2].

The interviews aimed to explore the concept of PD within each of the companies and what aspects impact and drive such debt (see Table I). 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-ups to learn about the aspects of the process, focusing on the characteristics described in Section 3.1.1 (according to our RQs).

Table I: questions asked in the first step of the study.

Question	Relation to RQ
Describe some critical sub-optimal processes	RQ1
What led the process to become suboptimal?	RQ2.1, RQ2.2
What are the consequences of such sub-optimality?	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

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 minutes 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 to 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.

Cases description

To provide more context for our study, this section describes the involved participants 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.

Participants

As illustrated in Table II, the interviewees work at companies operating in several different business domains, and they have several roles.

Table II. Participants in the first step of the study

Role	**Company**
Manager (Testing)	A
Process, Methods and Tools manager	
Software Architect	
Process, Methods and Tools manager	B
Process, Methods and Tools manager	
Senior Software Developer (Requirements)	
Process expert	C
Process, Methods and Tools manager	
Software Process manager	
Software architect	
Senior software developer (continuous integration)	
Software Architect manager (methodology)	
Senior Software Improvement manager	D
Technical manager	
Team Leader	
Senior Software developer	

3.1.3 Phase 3 – Analysis and Synthesis

Based on recommendations from Yin [1], the analysis was carried out using a code and pattern identification technique called explanation building to match the analysis from both the interviews and the documents into one, since 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 [1]. 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 Step 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 ([28]), and an analysis of the interviews conducted in the Agile program regarding teamwork and coordination in distributed development is reported in ([29]).

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 toward 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 (see Table III). We also observed 24 meetings and conducted more engaged research by conducting retrospectives with the teams, thereby assisting them in solving their difficulties with practical and theoretically grounded solutions. In addition, we collected documents such as plans, strategy reports, progress reports, evaluations, sketches and designs of systems.

Agile program

In 2017, Company E initiated an agile program which consisted of four cross-functional autonomous teams organized according to agile principles to develop software for their business-to-business insurance products. The teams included personnel from both the software development and business development departments and delivered software solutions to the business side of the organization, such as sales and settlements. The teams worked closely with the organizational departments responsible for innovation and technology development. 'We conducted ten interviews and facilitated three workshops. Additionally, we observed 15 meetings and collected various documents.

The interviews in the agile program lasted from 37 minutes up to an hour, with an average of 48 minutes. All participants agreed to be recorded, and the interviews were transcribed verbatim within a week of the final interview, which was conducted in January 2019.

Data collection and analysis – Company E

The interviews aimed to explore sub-optimal processes for working agile, for meetings, information flow and coordination. We investigated what could work better in the project and how they solved problems. We also asked about 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 sub-optimal processes and their consequences.

Table III: participants for step 2 of the study

Role	Program	Company
Project manager	Web	E
Manager		
Business developer		
Digital responsible		
Digital designer		
Team lead		
Tech lead		
Test lead		

IT developer Product owner 4 Developers Test Leader Business developer Business developer Enterprise architect UX designer	Agile

Our data analysis was guided by an interpretive approach that places practitioners' understanding of reality at the core of our study [22]. We subscribe to the concepts proposed by Klein and Myers [23], 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.

3.3 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 ten interviewees from four companies, while three researchers presented the results and asked additional questions. The total interview time was three hours.

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 sub-parts:

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 (RQ 2.3)

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) (RQ 3.2)

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 impactful consequences of PD (RQ 2.5).

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 first 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 RQ 2.3, 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 Process Debt definition (RQ1.1)

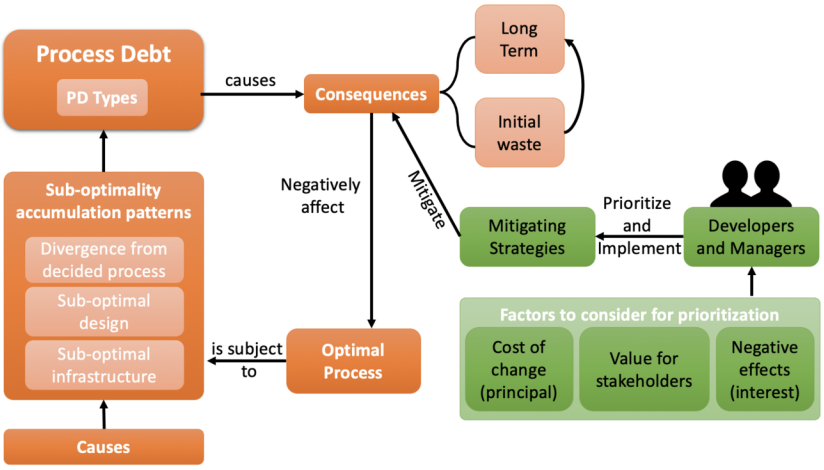

Figure 2. The overall framework of PD (orange) and mitigation strategies(green).

We propose an overall PD conceptual framework (Figure 2) 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. Also, a preliminary framework (without the mitigation strategies) is provided in our previous study [24].

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 a *theoretically optimal* one, 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 **sub-optimal** . 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 (discussed in section XXX). The starting point responsible for the accumulation of PD is either a process designer (who has created the process with a *sub-optimal design*) or a process executor (who has*diverged from the decided process*). In addition, the main source of PD can be a *sub-optimal infrastructure* . When a process emerges, often the process designer and executor can be the same, and in such cases, PD often occurs because of an *involuntary* sub-optimal design.

PD can be identified by a type: this can be a sub-optimal aspect of the process (role, documentation, synchronization with other processes etc.) or by a specific *sub-optimal activity* (this can depend on the specific activities that the process is composed of, for example is it a code review step, prioritization, etc.,). 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 mitigation strategy with respect to implementing other strategies instead or just continue developing 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's assume that the prioritization process could give a much higher value to the stakeholders than the review process, the former one should be prioritized.

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.

Validation:

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 [10] was based on the definition given for TD and we found several distinct differences between PD and TD, and therefore, 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 Process Debt (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, documentation, etc. 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, requirement debt etc.). Based on the results from our analysis the following types of debt were recognized (visible in Figure 3):

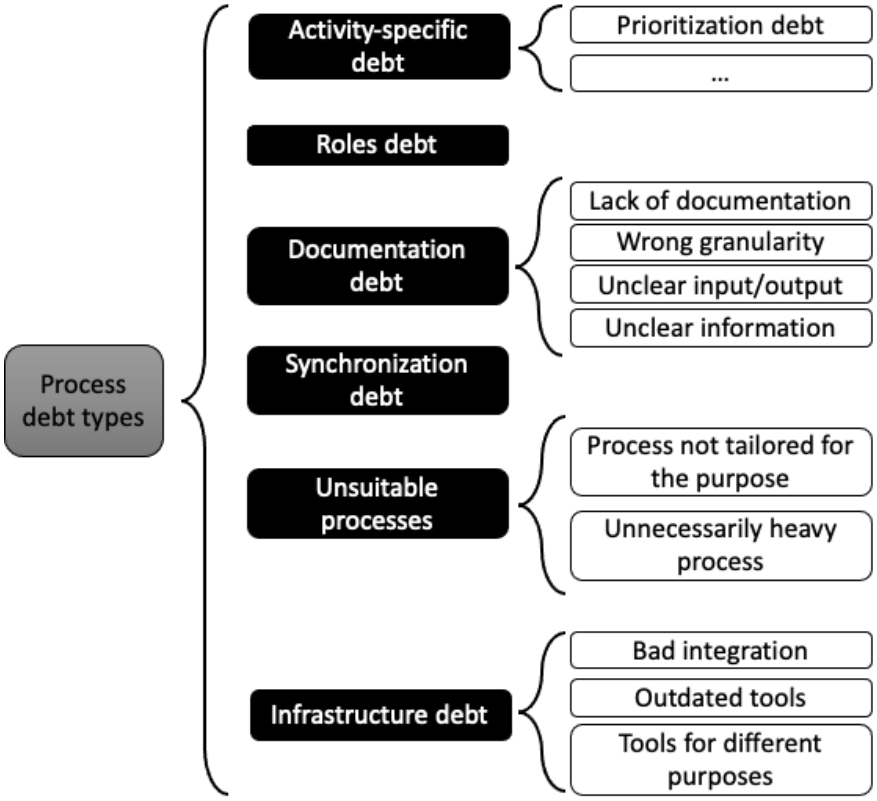

Figure 3. Different types of PD with sub-types

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 sub-optimal and not the whole process as a whole.

We found several kinds of activities that can be flawed: for example, a sub-optimal prioritization process could be called prioritization debt, a sub-optimal 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.

An example of an activity-specific debt was found in company D, where the main PD related to the entire bug-fixing process was specifically located in the use of an extensive checklist to be manually filled in by the developers every time a new bug was discovered and fixed (an instance of documentation debt). Such debt could be named by the specific sub-optimal documentation activity as "bug-fixing documentation debt".

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's 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, which 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 accepting and carrying out some tasks for which they are responsible in the process. For example, in Company E, they created a role that they called "product owner" and assigned people 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 was more related to being a team leader. Another issue is when responsibilities are not well mapped. An example from the agile program in Company E showed how 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.

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. To make a concrete example, company B explained that a process related to the versioning of the code was designed for hardware

stakeholders by stakeholders in the software domain. Some language was not clear, as some terms could actually have different meanings in software and hardware domains.

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, there might be a separate process that prescribes code reviews to ensure software quality and there might be 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.

An example that was mentioned by Company A was the presence of an overall certification process that coexisted together with Agile software development. The lack of a good synchronization across the two processes created frustration and inefficiencies with respect to both processes: teams might not be able to be agile, while the process occurred to have excluded some important certification documentation.

Another issue revealed by Company E, was that the offsite teams followed Scrum, while the onsite teams used Kanban. The different methods chosen affected how pull-requests were handled between the teams. In the Scrum teams, they sent pull requests to the teams onsite after the sprint. Meaning every second week a large number of pull requests were sent for review. It was seen as beneficial to use Scrum as it would shield the developers from having to review pull requests daily but instead do it more intensely over one week.

However, it was described as overwhelming for the onsite developers receiving the pull-requests, as illustrated by a developer: *"When I get QA from the teams following Scrum, then it is maybe 64 [pull requests], so then it's like "wow!" where should I begin? That is cumbersome, and not very motivating. Reviewing pull requests is never particularly fun regardless of who sent it, and now it has been quite a lot, then it gets really boring."* Further, it was commented that the off-site pull requests were more extensive (because they had been coding for two weeks) than the on-site pull-requests, which made the work even more demotivating as extensive pull request take more time and energy to approve, as it becomes more complicated.

In summary, the two processes were not well synchronized, creating PD. As a consequence, one of the teams was highly demotivated.

In general, stakeholders, besides being executors for a number of processes, also have a workflow (see section II.B). An example is a software developer that 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 sub-optimal workflow for the developers, where some activities are in conflict with each other. This is clear from company E, as explained above, and another example is given by company D: a developer mentioned that he had to get information from another unit, which was following a different process for defect handling. The received information had to be converted into a user story according to SCRUM. When the code was delivered for such a story, additional documentation was needed by yet another organization, which had the purpose of generating analytics. In addition, the developers' workflow contained other steps related to yet different processes, making their bug-fixing workflow continuously interrupted, making it tedious, error-prone and inefficient.

In conclusion, the main findings related to Synchronization Debt (PD) 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. Company A had in place a formal overall waterfall-like process for the development of their mechatronic product, while the software development teams were agile. The waterfall process was inherited from other disciplines, such as mechanical and electrical engineering. However, such a process was not suitable for software teams, which needed to develop software iteratively. In practice, informally, the software teams were working agile but they also needed to provide documentation to comply with the overall waterfall-like process. This caused confusion and inefficiencies, for example, when stakeholders needed information on a specific step of the process, but the documentation was not reflecting the activities that were carried out, which generated false information.

One more example is given by the longitudinal case-study: in company E, the business development unit followed a decision-making process that was not suitable for software development. Such a process was based on complex decision-making that involved continuous synchronizations and updates. This made the specifications change continuously. On the other hand, the software development team relied on Scrum, an iterative method in which the team began with a planning meeting and subsequently met daily for two to three weeks, presenting what they delivered at a demo meeting after that time. When software development started solving tasks that were put into a sprint (iteration), they soon discovered that the tasks were different (e.g., in terms of complexity, size, or the number of dependencies on other sub-systems) than what was originally planned for by the business development side.

"The decision-making process is a mess. For example, it is unclear to the developers who have the final decision on design and the development. Is it the product owner, the manager, is it the team in consensus or is it someone from the business? The consequence is an ineffective process where the developers believe decisions are made, and suddenly they are stopped in their development because it was not decided that that was the way to do it after all."

As a consequence, software development did not deliver what the business side had expected at the end of a sprint, causing business development to begin to lose trust in software development. Further, because the business developers were often not aligned or not available, software development seldom got fast feedback when they discovered that a feature or design needed to be changed, causing software development, in turn, to lose trust in the business side. The result of all these challenges was that the process from identifying a feature to delivering the feature took much longer than necessary.

The difference with synchronization debt is that, in the former case, the processes might be optimal but not well synchronized, while, for this category, there is one process that is just not suitable for the needs of (a part of) the stakeholders.

Main findings on Process Unsuitability:

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

Infrastructure debts (IDs) have been considered to be PDs by the interviewees. Here we explain in more detail what specific issues can be considered ID.

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 they 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. An example from company B is related to a software versioning tool that was purchased several years back and has

always worked well. However, when the company moved to continuous integration and continuous delivery, the tool did not support fast operation with the distributed codebase. Consequently, the process was slow, and developers would not deliver often to avoid additional overhead. The company decided to remove this PD and to purchase a new tool.

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 sub-optimal.

Finally, from the longitudinal study, Company E mentioned that people were forced to sit at specific desks two times a week as they belonged to two teams, which made people extremely frustrated because they wanted to sit at the same desk the whole week. Either they preferred to sit close to the people they were usually seated next to, or they relied on external monitors. In this case, it is clear that not only is it the virtual environment and tools used to develop software that constitutes PD, but infrastructure debt can also include the physical environment surrounding the practitioners.

4.3 Process Debt occurrence and its accumulation patterns (RQ2.1)

Three overall PD accumulation patterns were identified: Sub-optimal 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 Sub-optimal process design

This category includes those processes where a decision was made that caused the process to be sub-optimal 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. One example from company D was the need for a large number of fields in a report to be filled in by the developers every time a new release was delivered. This was decided by another organization, which used such information to compute analytics and statistics. When we investigated with the process designers and the developers, it was not clear why these fields were so important. They were decided a long time ago when releases were infrequent, and the developers did not have to fill in such parameters so often, and therefore it was not seen as a problem. However, as new processes (e.g., continuous delivery) were introduced to the software team, filling such parameters became a significant negative impact on the team. The emerging new processes had created an instance of PD by conflicting with the existing ones, and this was generating a substantial overhead.

Another example of sub-optimal process design was in Company E, where they decided that the Norwegian teams used Kanban while the outsourcing teams followed Scrum. The reason was that both sites could use the process they were familiar with and that they believed it was the most productive process. However, since they followed different processes, collaboration across sites became troublesome as activities were not synchronized and misunderstandings emerged. Also, several coordination meetings caused much overhead and waste.

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 sub-optimality, but for which a better tradeoff is not identified.

An example from company C was the formal process used in the company concerning the execution of unit tests. The process was designed upfront by the process owners, which were responsible for many of the processes in use at the company. The unit-test process included a step for which a certain coverage should have been achieved (a % of modules should have been tested) before delivery. However, in practice, most of the projects did not reach the recommended thresholds. According to the interviewees, this had an impact on the quality and reliability of the released software. In this case, although the unit-test process was well-defined, it was ignored by the development teams, causing an interest (in terms of quality and reliability) to be paid.

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 sub-categories: 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, infrastructure debt 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 (Processes, Methods, and Tools) units that need to coordinate processes and tools together. Given how intertwined the two categories are, it would not make much sense to separate infrastructure debt from PD. In practice, any infrastructure debt will also create a problem for the process. We refer to infrastructure deficiencies as infrastructure debt (ID, as part of PD) to denote that the tools are a source of the debt.

To mention an example from company D, a software process related to bug fixing originated from the fault handling being initiated by a unit developing hardware. The issues, created in a tool used by the hardware team, then needed to be processed also at a software level, and the software development team received a report from the hardware organization. The developers then needed to convert and duplicate the entry and add it to a software management tool (Jira). The lack of integration across the two tools created an unnecessary overhead for the developers every time they received a bug to fix. In particular, the interviewees mentioned that a solution would be possible to integrate the two tools. However, such development had been down prioritized by the management as it was not deemed important enough to implement.

Processes regarding the use of Jira also caused infrastructure debt occurrence in Company E. Since the offshore teams were measured by the number of solved Jira issues, it caused the developers to attach multiple Jira issues to one pull request to improve the performance measurements. The result was that the Norwegian developers had to review very large pull requests, which was frustrating for them as large pull requests became very complicated and took a long time to resolve. Reviewing small pull requests were perceived as much more efficient.

4.4 Causes of Process Debt (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 also illustrated in Figure 4.

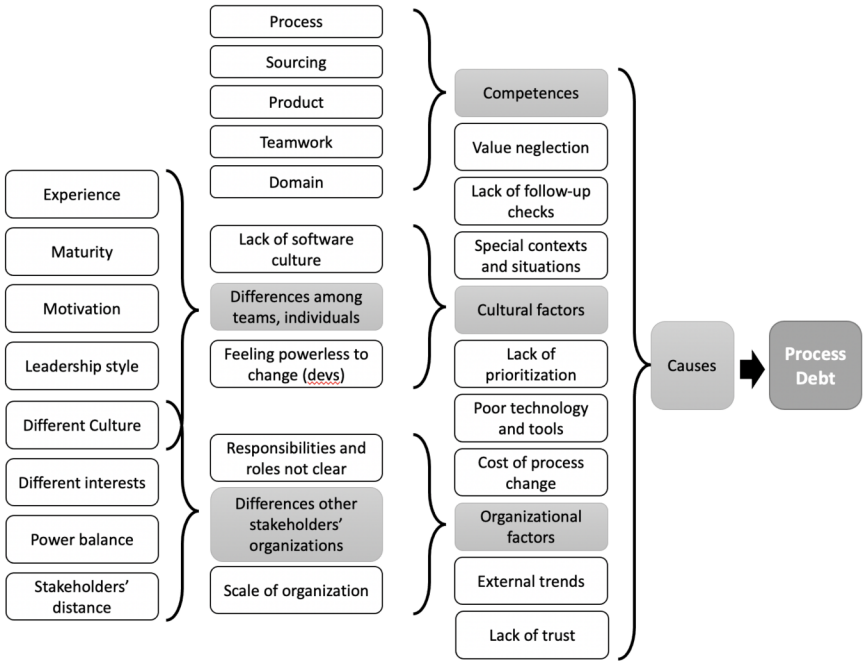

Figure 4. A map of the causes and consequences of PD. Dark grey boxes have sub-categories. "Different culture" is related to both teams and external stakeholders and therefore the inclusion brackets overlap.

1. **Competences:** An obvious issue leading to PD is the lack of competences to design, manage and execute *Processes* . Processes can emerge and not be designed upfront, but in any case it is important that processes are assessed, supported by infrastructure and maintained. As mentioned by an employee in company D, it is important to have a responsible person for managing processes and to have managers who understand the value of processes. 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. In company E, we found that the different departments had their own understanding of agile processes, which caused challenges when they collaborated across departments. Therefore, one crucial action they decided to take was to work on building process competence at the company level and share process knowledge across departments.
2. **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.
3. **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 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.
4. **Special contexts and situations:** different companies, units, teams, special domains, special events, etc. all contribute to making existing processes sub-optimal, which generates PD and costly consequences if such special contexts and situations are not accounted for in the process.
5. **Cultural causes:** there are several cultural issues that have been mentioned by the practitioners.

This category has been further elaborated in the following sub-categories:

6. A major issue is the **lack of software culture in other organizations** , for example, related to mechanical and electrical engineering. This means that processes that are well-designed for other contexts are not optimal for software development.
7. In addition, **different teams and different individuals** have different **experiences, maturity, motivation** and**leadership styles** (they might like or not to be guided by a process). This leads processes to be optimal for one team or one individual but not for another. Various issues have been mentioned in the interviews that are related to these cultural aspects: experienced people might neglect the process thinking that they do not need it (while occasionally missing newly added critical steps), while less experienced individuals and teams might not understand the purpose of the process and just rigidly follow it without judging special situations. The same process can be followed by individuals and teams who desire a proper guide to be followed, while others might instead desire to be only self-guided and independent (even at the expense of other organizations).
8. **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.

One of the interviewees from company A explicitly estimated that cultural issues could even account for **a third** of the total number of PDs within their company.

1. **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 responsible for implementing or changing the process.
2. **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.
3. **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, company C mentioned that 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.
4. **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 very different organizations with different **cultures**, **interests** and**power**: examples are processes for example (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, or yet again organizations that are just used to employ unnecessarily heavy

processes, which make software development inefficient. The issue is not so much the existence of external organizations, but rather the**distance between the stakeholders** in the software development team and such organizations. It is possible that the different organizations, to communicate, have to go through several levels of indirection (talking to a manager who talks to another manager etc.), which does not contribute to making the stakeholders aware of each other 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 followup 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, especially from company A, 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. For example, in Company E, the interviewees described that some developers were reluctant to share knowledge and their competence. One explained that sharing his expert knowledge could make him less important to the company and that others could take over his job. Another explained that developers resisted letting others access the source code as they did not trust them to do a good job with it.

4.5 Which PD causes are the most common? (RQ2.3)

From Figure 5, it is possible to see how the participants from the cross-company interview chose the causes that the most were causing PD (RQ2.3). Three voters for company A, two for company B, 3 for company C and 2 for company D.

The most voted, and the one that was voted by all the participating companies is the presence of **poor technology and tools** . The second most voted one, especially from company A, was the**neglect of the process value** . **Special context and solution** and **Lack of improvement prioritization** were voted on by three organizations.

However, it is interesting to see how different roles and participants from different companies have voted differently. This might mean that in different sub-contexts, different causes might generate more PD. All in all, one could consider most of the causes as contributing to generating PD.

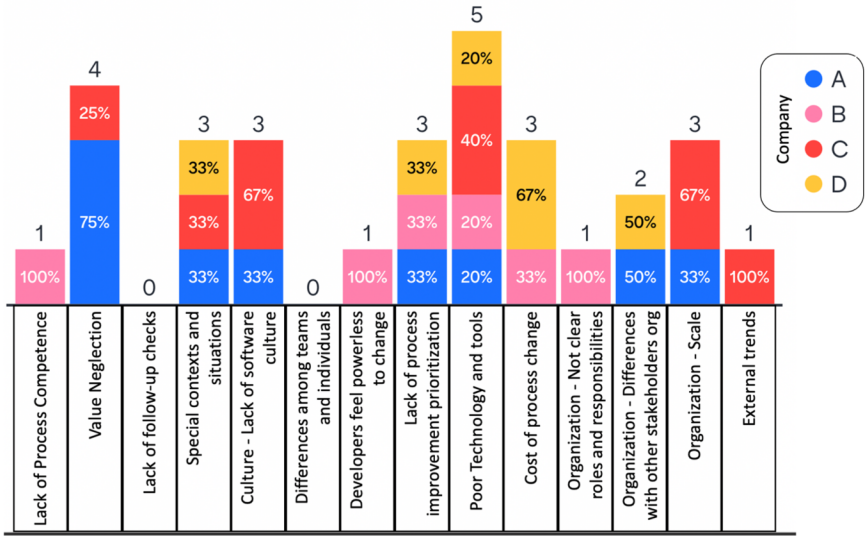

Figure 5. The causes that were voted as the ones causing the most PD in the organizations. Respondents had to pick the three most important.

4.6 Consequences of Process Debt (RQ2.4)

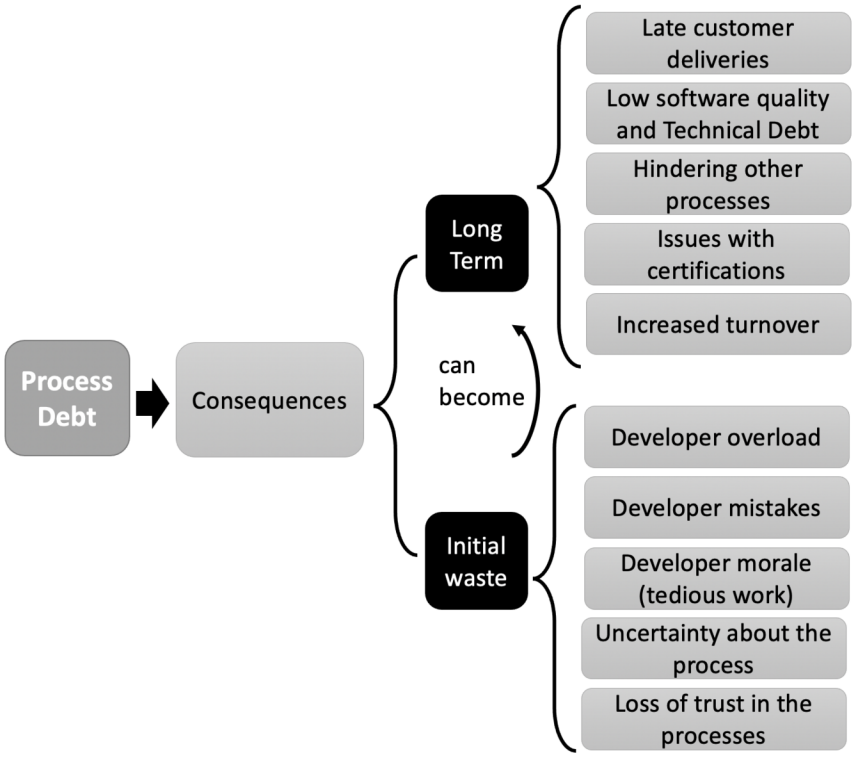

Figure 6. Consequences of PD

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 sub-optimality 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 commonly such a way of working would cost more in the long run and could also cause longer delivery times in the long term. In some reported cases, the customers were not aware of the PD and how it was accumulated and its consequences, which made them unhappy when the unexpected long-term delays showed up.
2. **Low software product quality and TD:** one of the purposes of processes is to assure that products are delivered to the customers and are satisfying several qualities critical for the customers. Many PD instances have been mentioned by the interviewees as generating quality issues in the long term. For example, not delivering proper coverage for unit testing can create additional defects in the long run, and can undermine the confidence of the organization in delivering additional features that are not appropriately tested (company C). Another example is given by several of the companies, who mentioned how PD quite often generates TD. For example, sub-optimal code reviews do not catch TD in the code, while the lack of a process to manage the iterative and continuous feedback loop between software architects and development teams can lead to architecture debt.
3. **Hindering other processes:** processes are often intertwined, and the presence of PD in some processes might often generate disruption in other processes as well, generating new PD (e.g., in the case of synchronization debt). This is a characteristic that leads to the contagiousness of PD and possible vicious circles. An example mentioned by our interviewees was related to an existing PD (infrastructure debt) related to an outdated tool whose technical issues prevented the implementation of continuous integration.
4. **Issues with certification:** another example reported by companies A and D was related to skipping process steps related to certifications. Such steps often provide documentation and evidence that the product has indeed successfully gone through all the steps that are required by the certification authorities. In the end, this could cause difficulties in obtaining certifications or the need to re-validate and re-perform steps of the process wasting a large amount of time, resources and budget in the long term.
5. **Increased turnover:** In Company E, some people were frustrated by the PD issues related to the incompatible processes between the on-site and off-site teams, resulting in a higher attrition rate. Especially lack of competence among the off-site team members caused frustration. Further, role debt in the case of the product owner and team lead in Company E caused people to quit because they did not understand their responsibility, authority and how to best collaborate and coordinate with other team members and stakeholders.

4.6.2 Initial waste

When process designers create a process, it can be the case that the process is sub-optimal right from the beginning. This can be due to many challenges, such as, 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. Sub-optimal documentation granularity, infrastructure debt, the presence of unsuitable processes, and other activity-specific debt were the source of

unnecessary effort spent by the developers in activities that were sub-optimally designed as mentioned for several instances presented in section IV.B.

2. **Developers' mistakes:** some PD affects the developers by making their workflow error-prone. For example, a developer from company D mentioned how the frequent copy-paste and duplication of information from one tool to another (because of infrastructure debt) caused him to make mistakes that were 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. In companies B and D, where we were able to visualize the studied process, the interviewees mentioned how they were helped in catching the presence of PD.
3. **Developers' morale (tedious work):** one issue, especially related to documentation debt and infrastructure debt 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 sub-optimal 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 example, in Company E, the product owners 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 product owners 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.
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 sub-culture 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 impactful? (RQ2.5)

From the cross-company interview, we collected data as in Figure 7 Three voters for company A, two for company B, 3 for company C and 2 for company D. The most impactful PD consequence was considered the one generating **low software quality and technical debt.** Other voted by four or five participants, and generally voted by more than two companies, are the **uncertainty about the process,** the**loss of trust in the processes** and the impact on**developers** (**morale** and **overload**).

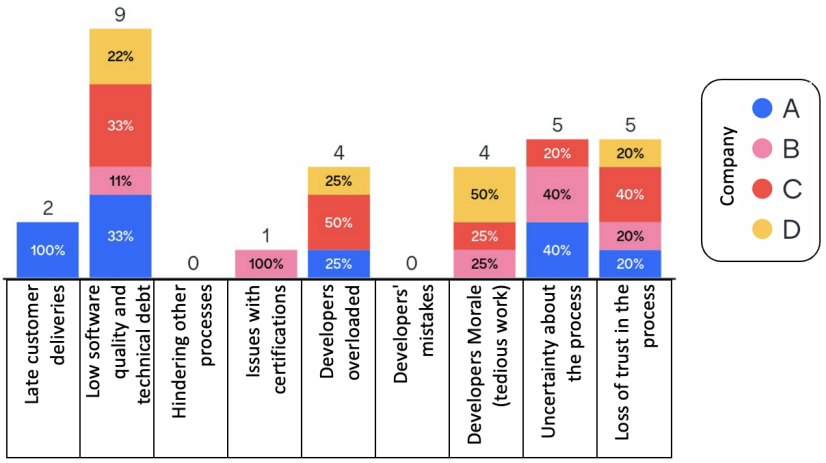

Figure 7. The consequences of PD that were voted as the most impactful by the participants

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 step 1 and step 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)

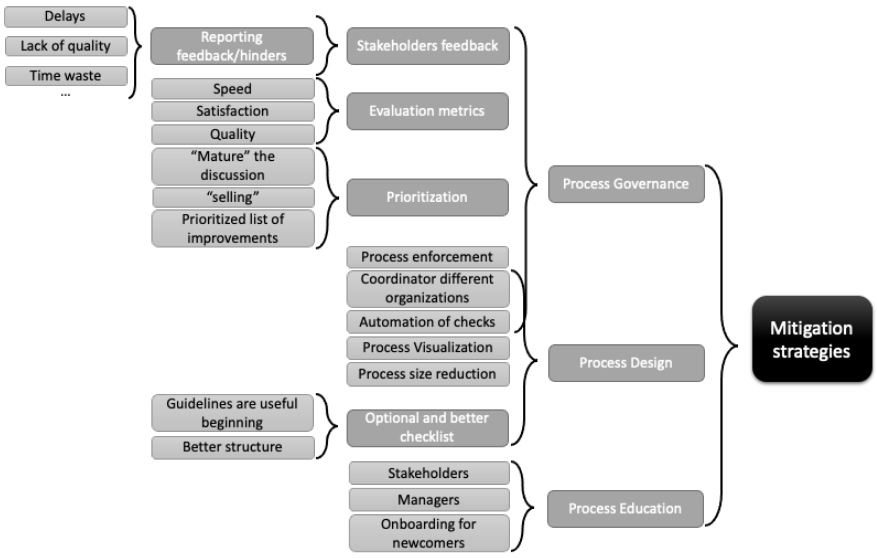

Figure 8 Mitigation strategies proposed by the interviewees. Dark grey boxes have sub-categories.

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.

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.

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 **hinders** 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 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. As one of the respondents from company D mentioned: *"I think it has originated from us developers pushing on very much [. . .] we're using way too much time on this, and then at some point it gets too much, and then the management hears it [we reported] both the amount of time that it takes and the amount of time that it delays our releases" "I remember at least one of the optimizations that were done. The way it got approved was that some of the other managers had to write "we do this perhaps 200 times a year, and we'll save two hours every time." Suddenly 400 developer hours. Then it's suddenly justifiable."***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 mentioned that current approaches are rather inefficient and better approaches need to be developed.**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 in 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 the 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, technical debt, bug fixing, social issues, etc.).**Process enforcement**: Some of the practitioners mentioned how enforcing the process with checks and tools can help reduce and avoid PD. For example, when referring to an obligatory form to be filled in during the process to comply with an external certification process, one of the interviewees from company D mentioned: *"We've made the system in a way where we can't really ignore them [data to be filled in during the process]. We have to fill them. And I think that's actually quite good, because otherwise we would close the tasks and then when we hit the release date, then we discover, okay, half the things haven't been filled out, and then it's a huge task to fill in the data.***Coordinator for different organizations:** when different organizations require a process to be in place for the software teams (e.g. for safety certification, analytics etc.), it is important to have an effective coordinator to bridge the needs of the two sides and to foster their communication. For example, company C managed to conduct a comprehensive effort (managed by an appointed coordinator) to share the same language between the software and the hardware units leveraging system engineering practices, while company D mentioned that they are in the process of appointing a coordinator between the software teams and the analytics unit to improve the reporting process, discussing what additional information is really needed by the software developers and which one is not and can be removed from the mandatory documentation. This strategy overlaps with *process design*, as the coordinator can be a role that is included in the role definition of the process.**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. *"A lot of these [manual steps in the process], almost all of the things that I find annoying here could be automated. Definitely."*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 a governance strategy, but can be part of the process design overall category, as their definition can be included in the design of a process.

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.

Process visualization: one of the ways to identify PD and to argue for its reduction is to visualize the process. Company D also showed us an instance of the discussed process that was reproduced by one of the developers that were skilled with visualization techniques. In such an illustration, it was quite clear where the bottlenecks were happening (for example where a manual copy-paste from two different tools was performed). According to the participants, presenting the visualization to the managers convinced them that the PD was detrimental and that it needed to be tackled (for example by introducing automation between the two tools). Process visualization can also include data-driven approaches. However, this doesn't seem to be currently an approach that is in use at the interviewed organizations.**Process size reduction**: some of the companies, for example A and B, are very large organizations including several disciplines and are characterized by the need of having formal processes that, in some cases, are pre-defined. In other words, some steps that are included in the processes are there because the process has been standardized: for example, a process to ensure that unit tests reach a given coverage might be more suitable for some teams but not others. Company B mentioned that such 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 are not revised to be improved, and, in particular, simplified, for example, by removing obsolete steps.**Optional and better checklists:** to help executors follow the processes, often checklists with 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. *"So when people are learning and giving feedback [. . .] so that we actually can update when we find better ways of doing things [process checklists]."*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 learnt 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, can therefore reduce the overhead of the teams and still allow practitioners to be reminded of critical steps that should not be overlooked.**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.

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. For these reasons, education is massively used by company A:

"The process here is quite complex, and there are a few other teams that see the value of how to do it. [. . .] We have to educate people. If we have something that people don't understand, then we have to have an education that explains it.". In Company E, they held internal workshops to get a common understanding

of roles and processes, and cross-department workshops on agile culture. In addition, they established cross-company guilds on agile methods for all departments, not only the software development department.

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. Figure 9a and 9b 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** (7 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 2 answers (used today) to 7 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.

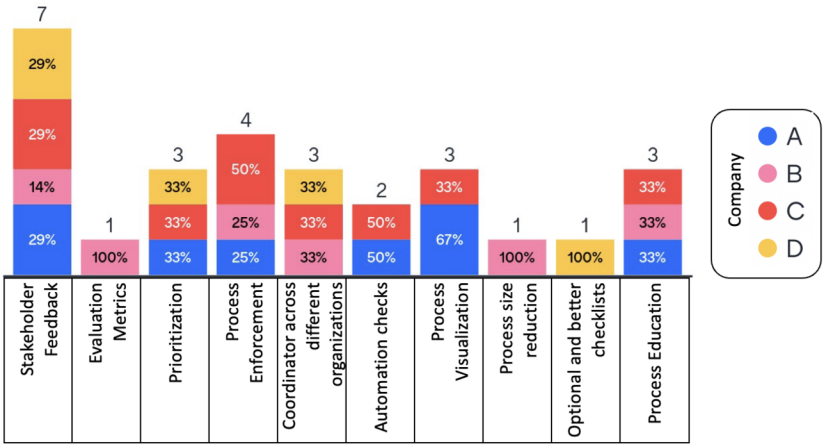

Figure 9a. Mitigation strategies mostly used in current practices

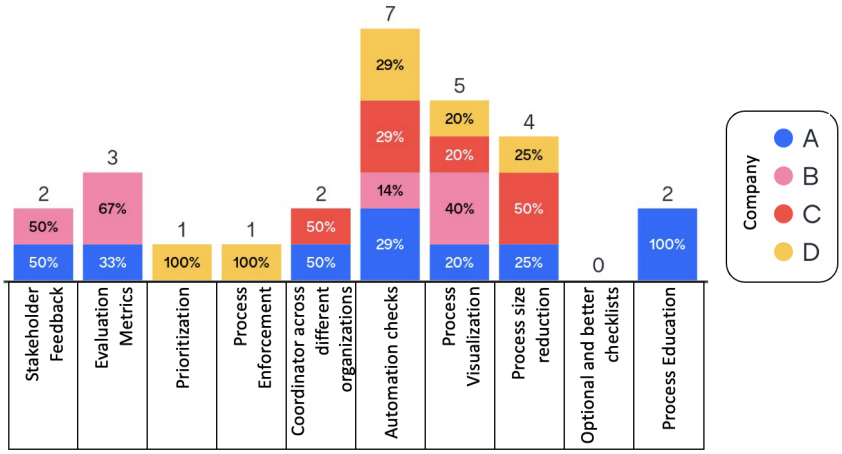

Figure 9b. Mitigation strategies that are wished to be applied in the future by the participants

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: The introduction of processes like DevOps and BizDev was implemented because of a need to reduce the problem of disconnected activities and become more customer-driven. As written in an internal document, "*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".* When traditionally separated processes become continuous, new processes replace the existing processes to connect planning, testing, and development activities. Disconnected processes are examples of synchronization debt and unsuitable processes not tailored to the purpose of the companies and the teams. Such debts emerge because of the need to deliver faster.

At the beginning of the study in Company E, business development and software development were separated, including the organizing principle that business developers prioritized what the software developers should deliver without involving them. Consequently, both the business and the development sides optimized their processes. However, running two different processes caused synchronization debt. Software development did not consistently deliver what the business side had expected, and the business side started projects that could not be developed from a technical viewpoint. Further, because the business developers were hard to reach, software developers seldom received fast feedback when they discovered that a feature or design needed to be changed, causing the time from identifying a feature to delivering the feature to take much longer than necessary.

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.

"*We make decisions along the way, while we work on our tasks, when it is needed. If there is something under development, and an issue occurs, it happens that we deal with it informally by talking with the developers about solving it differently."*

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 set-up increased the speed of feedback and clarifications, frequent interruptions made the developers' day fragmented, 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, as the teams were large (13-20 people), and consisted of people with different roles and tasks, some of the meetings (e.g., the standup meeting) did not give value, as what people discussed was not relevant to everyone. This meeting set-up caused reduced motivation because the meeting attendees felt that the time spent in the meetings was a waste of time. To mitigate the problem of having too large teams and one continuous process, the teams were later split into sub-teams with dedicated responsibilities.

Redesigning processes: However, they still had handover problems between testers and developers. The developers wrote the code, and the testers tested the code. This process was suboptimal because it caused delays and misunderstandings (classical problems). To mitigate this PD, a new process was introduced. They introduced a pull request (PR) approach so that the developers could assign people to check the code written, which made it easier to onboard people. Eventually, they removed the tester role and gave everyone responsibility for quality.

4.9.2 Sourcing, competence, and unsuitable, continuous processes reveals PD

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. Since they followed different processes, it was discussed if the code review practice should be continuous or timeboxed.

Sub-optimal 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 Qas and reducing the number of interruptions for the Norwegian developers. This process turned out to be sub-optimal. Every second week, because of weak domain competence at the offshore team, the Norwegian developers received large pieces of code. Since 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.

The PRs were not only large but also of low quality. The two sites did not follow the same standards and structure of coding. Also, much code was copied from other projects or the internet, which meant there were pieces of code not being used or irrelevant to the project. Unused code was described as adding unnecessary complexity. Another problem was that the offshore developers tended to overly complicate the code of relatively easy tasks. These problems created tension between the teams.

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."

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 (section 4.8.1), they can also be misused, 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 Jiraissues, to double the KPI. I think measuring an outsourcing partnership based on the number of approved PRs is not a good solution."

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 the communication more informal and faster and, therefore, more quickly could clarify issues related to PRs. The situation improved a bit.

However, it became clear the new process did not work out as intended after a while since the quality problems persisted. Further, the remote company had a high attrition rate, resulting in new experienced developers replacing the people who could deliver appropriate code quality after a year. The annual turnover rates were: 2016: 4%, 2017: 18%, and 2018: 38%. The new people joining the company in India did not have the proper technical skills, domain knowledge or process knowledge. As a consequence of low productivity and processes that did not work, the sourcing relationship was eventually canceled for the teams.

4.9.3 New roles and roles debt

In the longitudinal study, we saw all types of PD. PDs were mainly identified and discussed through retrospectives in the company and better understood in the interviews.

Roles debt: Synchronization and alignment debt were evident early in the study and were problems that existed over several years. For example, in Company E, they introduced a role that they called "product owner" and assigned people to that role without explaining or describing what the role entailed in detail. It was believed that a short memo was enough. The product owner's responsibilities were mainly related to being a team leader. However, many of the people assigned to that role were confused, as some were familiar with the product owner role in Scrum. Further, as there were many POs, they needed to collaborate. But 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 action points were specified.

A new description was created explaining that the role had responsibility for the team, prioritizing the business needs and coordinating with the stakeholders. However, even though the role was more precise, the work was difficult to perform because an enormous amount of coordination and communication was required for the POs. Consequently, a new role was introduced – a coordinator to coordinate between the POs and the management. This person was to represent the POs in management meetings. The effect was supposed to be less work and less coordination for the POs. However, as the POs they were taken out of important decision processes, their work became slower as decisions took a long time and several misunderstandings happened.

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 the POs became more and more experienced (after a year), the process was modified, and the POs were given more and more authority and could make more decisions without having to ask others.

Still, the process was flawed. The POs needed to coordinate with many stakeholders. The reason was to make sure all stakeholders were informed, involved, and committed to the work performed. While the effect of the process and roles were ok, the work took too much time. The solution was then to invite all key stakeholders to the coordination meeting for the program. Instead of POs reaching out and informing others, the key stakeholders had to come to them, which worked much better.

5. DISCUSSION

5.1 Contributions and implications

Process Debt is a phenomenon that has not been studied as much as Technical Debt. 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 Process Debt

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 systematically manage PD and need a comprehensive reference point. First and foremost, TD research has shown how 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 Software Process Improvement 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 Process Debt state of practice

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

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 mentioned 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 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 be able to reason about prioritizing PDs' 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 often repaying TD implies that the new solution is without better (excluding a few corner cases). This means that evaluating if repaying PD is worth it, a more complex assessment is needed, 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 Schön [21] and the use of double-loop learning, where additional evaluation is done after a solution is applied.

The most valuable part of our reported PD state of practice is perhaps related to the reported mitigation strategies used by the practitioners to manage PD, which can also be applied in other contexts (if not precisely, at least can give inspiration for new practices). In addition, 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 to researchers to investigate novel approaches along the directions mentioned by our informants.

5.1.3 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 [25]. 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, for example, some of the causes and consequences might not apply or be different in such contexts. We, therefore, acknowledge that further research is required. However, we argue that our exploratory effort is already valuable, in that we included different companies with different contexts and different stakeholders, ranging from process designers to software developers, to 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 by running a workshop and 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, but we call for the software engineering community to contribute to studying PD with additional investigations.

As for reliability, although it's often the case that qualitative studies rely heavily on the researchers' interpretation, we have used several mitigation strategies to limit the bias introduced by the authors. First, we made sure that in the majority, at least two researchers were present at the interviews, and two researchers analyzed the interviews in parallel and then discussed and agreed on the resulting coding. At least two researchers

have discussed the findings and definitions across studies to ensure that the 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 run in phase 3 from the practitioners in the longitudinal case study related to phase 2, but only related to 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 Process Debt in five companies in three steps, including interviews, workshops, direct observations over a longitudinal case and cross-case validation comparisons.

Current state of the art and practice of PD is lacking a solid framework to reason about the concept and empirical evidence of the state of 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 Process Debt and its empirically-based state of art and practice concepts. Our contributions offer a comprehensive approach to understanding PD across domains and provide a starting point for software organizations to manage the phenomenon efficiently. By understanding how different parts of software organizations can manage PD and its effects, we articulate a strategy that allows software organizations to manage PD efficiently. We provide an initial framework, including a definition, taxonomies and an overall conceptual model, representing a novel reference point for practitioners to understand, reason about and manage PD.

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 make connections between different PD processes, identify commonalities among them and thus better comprehend the complex PD landscape. Furthermore, the framework can be used to analyse and structure PD initiatives, suggest improvements, and provide guidance for decision-making. It can also be considered the foundation to support further research: PD is a harmful phenomenon affecting software products and practitioners: it needs attention in software organizations and, consequently, in academia with empirical investigation and additional practices. We report a state of practice including concrete 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 reasoning about the prioritization of process improvements concerning competing activities such as feature development. We argue that some existing TD practices can be adapted to help with the management of PD, while new approaches are needed to tackle specific issues, as processes have many aspects that might not concern the management of TD.

We suggest that Process Debt can be managed by recognizing it as such, formalizing it, prioritizing its resolution, and using relevant metrics to keep track of its progress. We emphasize the need for collaboration across organizational and team boundaries to ensure that Process Debt is properly identified and managed.

With this first comprehensive study, 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. We seek to continue a dialogue to identify and discuss the specifics of the diagnoses and prescriptions for better Process Debt management and to raise awareness of the issue in the wider community. We hope our work will provide a useful starting point for further research, theory and collaboration.

ACKNOWLEDGMENT

We thank the interviewees from the Software Center companies (company A-D) for the invaluable insights provided as well as the participants in company E.

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, vol. 85, no. 6, pp. 1213–1221, Jun. 2012

[2] O. Pedreira, M. Piattini, M. R. Luaces, and N. R. Brisaboa, "A systematic review of software process tailoring," SIGSOFT Softw. Eng. Notes, vol. 32, no. 3, pp. 1–6, May 2007, doi: 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, vol. 26, no. 4, pp. 536–554, Jul. 2008

[4] Software Engineering Institute, "CMMI for Development, Version 1.3," Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2010-TR-033, 2010. Accessed: Oct. 09, 2020.

[5] S. Zopf, "Success factors for globally distributed projects," Software Process: Improvement and Practice, 2009.

[6] D. Moitra, "Managing change for software process improvement initiatives: a practical experience-based approach," Softw. Process: Improve. Pract., vol. 4, no. 4, pp. 199–207, Dec. 1998

[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, vol. 156, pp. 41–61, Oct. 2019, doi: 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, vol. 167, p. 110586, Sep. 2020, doi: 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, vol. 101, pp. 193–220, Mar. 2015, doi: 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, Cham, 2019, pp. 112–119, doi: 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),"."

[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), May 2013, pp. 93–96, doi: 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, vol. 29, no. 2, pp. 419–443, May 2018, doi: 10.1287/isre.2018.0788.

[14] A. Martini and J. Bosch, "An empirically developed method to aid decisions on architectural technical debt refactoring: AnaConDebt," 2016, pp. 31–40, doi: 10.1145/2889160.2889224.

[15] I. Sommerville, Software Engineering. Pearson Education, 2007.

[16] P. Veyrat, "What are the differences between workflow and processes?," HEFLO BPM, Jan. 30, 2018. https://www.heflo.com/blog/bpm/workflow-and-processes/ (accessed Jul. 15, 2020).

[17] Küpper, Steffen, et alt. "How has SPI changed in times of agile development? Results from a multi-method study." Journal of software: evolution and process 31.11 (2019): e2182.

[18] Kuhrmann, Marco, Philipp Diebold, and Jurgen Munch. "Software process improvement: a systematic mapping study on the state of the alt." PeerJ Computer Science 2 (2016): e62.

[19] Clarke, Paul, and Rory V. O'Connor. "The influence of SPI on business success in software SMEs: An empirical study." Journal of Systems and Software 85.10 (2012): 2356-2367.

[20] Moe, Nils Brede. "Key challenges of improving agile teamwork." International conference on agile software development. Springer, Berlin, Heidelberg, 2013.

[21] Argyris, Chris and Schon, Donald A., Organizational Learning II: theory, method and practice. Reading, Mass.: Addison-Wesley. (1996).

[22] Walsham, G. 1995. "Interpretive Case Studies in IS Research: Nature and Method," European Journal of Information Systems, (4:2), pp. 74-81.

[23] Klein, H. K., and Myers, M. D. 1999. "A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems," MIS Quarterly (23:1), pp. 67-93.

[24] A. Martini, T. Besker and J. Bosch, "Process Debt: a First Exploration," 2020 27th Asia-Pacific Software Engineering Conference (APSEC), 2020, pp. 316-325

[25] Runeson, P., Host, M. Guidelines for conducting and reporting case study research in software engineering. Empir Software Eng 14, 131 (2009). https://doi.org/10.1007/s10664-008-9102-8

[26] M. Unterkalmsteiner, T. Gorschek, A. K. M. M. Islam, C. K. Cheng, R. B. Permadi and R. Feld", "Evaluation and Measurement of Software Process Improvement—A Systematic Literature Revi"w," in IEEE Transactions on Software Engineering, vol. 38, no. 2, pp. 398-424, March-April 2012, doi: 10.1109/TSE.2011.26.

[27] an Solingen, R., Berghout, E., Kusters, R., & Trienekens, J. (2000). From process improvement to people improvement: enabling learning in software development. Information and Software Technology, 42(14), 965-971.

[28] Mikalsen, Marius, et al. "Agile digital transformation: a case study of interdependencies." Proceedings of the 39th International Conference on Information Systems (ICIS). Association for Information Systems (AIS), 2018.

[29] Stray, Viktoria, et al. "An empirical investigation of pull requests in partially distributed BizDevOps teams." 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.