Aalborg Universitet

AALBORG UNIVERSITY

The Journey to Technical Excellence in Agile Software Development

Alami, Adam; Krancher, Oliver; Paasivaara, Maria

Published in:
Information and Software Technology

DOI (link to publication from Publisher):
10.1016/j.infsof.2022.106959

Creative Commons License
CC BY 4.0

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

Citation for published version (APA):
Alami, A., Krancher, O., & Paasivaara, M. (2022). The Journey to Technical Excellence in Agile Software Development. *Information and Software Technology*, *150*(106959), Article 106959. https://doi.org/10.1016/j.infsof.2022.106959

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

Take down policy
If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

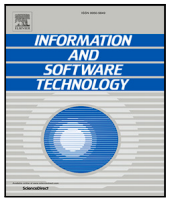
Downloaded from vbn.aau.dk on: December 06, 2025

Information and Software Technology 150 (2022) 106959

Contents lists available at ScienceDirect

Information and Software Technology

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



The journey to technical excellence in agile software development

Adam Alami [a,*], Oliver Krancher [b], Maria Paasivaara [c,b,d]

[a] Aalborg University, Denmark
[b] IT University of Copenhagen, Denmark
[c] LUT University, Finland
[d] Aalto University, Finland

ARTICLE INFO

Keywords:
Agile software development
Software development methods
Technical excellence
Agile principles

ABSTRACT

Context: Technical excellence is a nebulous term in agile software development. This vagueness is risky because it may lead to misunderstandings and to agile implementations that may overlook a key principle of agile development.
Objective: This study investigates how agile practitioners interpret the concept of technical excellence brought up in Principle 9 of the Agile manifesto. Moreover, we investigate how agile practitioners put the concept into practice and what conditions facilitate putting technical excellence into practice.
Methods: We conducted semi-structured interviews with twenty agile practitioners, coded the data inductively, and performed two sessions to validate the emerging findings.
Results: We find that technical excellence is first and foremost a mindset that is underpinned by continuous attention to sustainable code, continuous learning, and teamwork. Fostering technical excellence requires the adoption of design and development practices, such as continuous architecting, and is supported by continuous learning. We also identify three enabling conditions for technical excellence: Leadership support, customer buy-in, and psychological safety. These enablers provide teams with leeway to nurture their pursuit of technical excellence.
Conclusion: Our findings highlight the key role of people-based strategies in promoting technical excellence in agile software development. They show that the attainment of technical excellence does not only involve technical practices. On the contrary, it relies on social and organizational support and, most importantly, a mindset.

1. Introduction

Agile software development methods such as Scrum and XP are now widely adopted in the industry [1,2]. These methods aim at improving a software development team's agility (i.e., its ability to create and respond to change) by relying on iterative development, self-organizing teams, craftsmanship, and processes that are light and maneuverable but provide sufficient coordination for project behaviors [1,3–5].

At the heart of the agile movement are twelve principles outlined in the Agile Manifesto [6]. For example, Principle 1, *"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."* [6], emphasizes satisfying the customer through continuous software delivery, while Principle 2, *"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage"*. [6], welcomes changing requirements [6]. One common criticism of these principles is that they are too vague [2,7], which may lead to two undesired outcomes. First, teams may use agile methods without sufficiently understanding the manifesto's underlying values and principles, believing that the mere adoption of recommended practices enhances agility [2]. Second, different agile practitioners[1] may mean different things when they refer to agile and its principles. This may lead to misunderstandings and even to poor implementations of agile methods [2]. Therefore, an important task for teams is to establish a shared understanding of agile principles early in the development process [7]. In a similar vein, as Dingsøyr et al. [8] concluded in their review of the agile software development literature, an important mission for research is to clarify the "core" of agile values and principles [8].

> ***Principle 9:***
>
> *"Continuous attention to technical excellence and good design enhances agility"*. [6]

* Correspondence to: Aalborg University, Department of Computer Science, Selma Lagerlöfs Vej 300, DK-9220 Aalborg, Denmark.
E-mail address: adaa@itu.dk (A. Alami).

[1] For brevity, we use the term *agile practitioners* to refer to professionals working with agile software development methods.

https://doi.org/10.1016/j.infsof.2022.106959
Received 13 November 2021; Received in revised form 18 May 2022; Accepted 19 May 2022
Available online 30 May 2022
0950-5849/© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Following these calls to clarify key agile principles, this paper focuses on *Principle 9* (stated above), which emphasizes technical excellence and good design. A focus on *Principle 9* is relevant for three reasons. First, even though technical excellence and good design are often seen as critical issues in agile development [4,9], the terms are not defined in the Manifesto. This leaves agile practitioners with substantial room to interpret what technical excellence means and how it can be achieved. Second, while software engineering research has explored many issues related to agile development [8], there is surprisingly little work on technical excellence. Existing work is either practitioner literature (e.g., [10–12]) or not empirically founded (e.g., [13]. Thus there is limited knowledge not only about the interpretations of technical excellence among agile practitioners but also about the practices that developers and teams can rely on to achieve technical excellence and about the organizational conditions that support teams in their quest for technical excellence. Third, while other principles (e.g., welcome changing requirements) denoted a significant departure from the plan-based methods advocated before the agile movement, issues of design and excellence have also been a cornerstone of plan-based software development [14]. Indeed vast bodies of research on software architecture [15] and process improvement [16,17] have focused on issues of design and excellence. Against this backdrop, our focus on *Principle 9* promises insights into the specific understanding of technical excellence and good design that have emerged in the agile practitioner community. We address the following research questions:

RQ1: *How do agile practitioners interpret Principle 9 of the agile manifesto?*

RQ2: *Through which practices do agile software developers and their teams foster technical excellence?*

RQ3: *What are the enabling conditions for technical excellence in agile software development teams?*

We find that agile practitioners interpret technical excellence as a mindset that emphasizes continuous attention to sustainable code, to learning, and to teamwork. This mindset underpins the design and development practices that aim at achieving technical excellence and the continuous learning practices that help improve design and development practices. Key enabling conditions are leadership support, customer buy-in, and psychological safety, all of which provide leeway for developers to invest time in architecting and learning. Based on these findings, we propose guidelines for developers, teams, and organizations to operationalize the term technical excellence, put it into practice, and create conditions that enable teams to cater for technical excellence.

Our paper makes several contributions. To the best of our knowledge, our work is the first to trace current industry practices back to *Principle 9* of the agile manifesto. It establishes a chain of reasoning from a practitioner perspective and consequent implications, going beyond existing work on technical excellence that is either practitioner literature [10–12] or not empirical [13]. Moreover, to the best of our knowledge, our paper is the first to examine enabling conditions for technical excellence. Our paper also extends research on software architecture and process improvement, which has rarely looked at the interaction of architecting and process improvement practices and on the role of mindsets. Our study shows how architecting and process improvement work together for technical excellence and how a mindset can enable teams to care about technical excellence, even without formal processes and dedicated architects. An earlier version of this work [18] examined technical excellence focusing on **RQ1** and **RQ2**. This extended version adds **RQ3** with a revised analysis of **RQ1** and **RQ2** and enhanced presentation of all RQs.

We present and contrast our contribution with the related work in Section 2. Section 3 presents the study design and analysis methods. Section 4 provides the key findings. We discuss our findings in relation to previous work in Section 5. Then, we propose potential implications on practice and research in Section 6. We conclude in Section 7.

2. Background and related work

Our search of agile software development literature shows a mounting interest in various topics related to quality in agile, but the term technical excellence is rarely used in the literature. The results of our search show no available work directly linked to *Principle 9* of the manifesto. The available literature investigates various topics in isolation, such as pair programming (e.g., [19]), refactoring (e.g., [20]) and test-driven development (e.g., [21]). While these practices are popularized by agile implementations, they do not necessarily imply that agile teams pursue and sustain technical excellence.

Few literature streams are relevant to our work. While software craftsmanship is regarded as a feature of excellence, other streams such as quality in agile are connected to our work. In this review of related work, we present and discuss mainly software craftsmanship, software quality in agile and their relevance to our work, and the call for further work on technical excellence; then we briefly discuss how software architecture and software process improvement relate to technical excellence.

2.1. Software craftsmanship

An important fact is that people write software. As discussed by Pyritz [13], an excellent architecture, model, or process cannot, in itself, produce high-quality software — this work requires highly skilled software craftsmen who create with skill and dexterity. The craft of software transcends the technology curve; technologies will come and go, but the essential skills and wisdom of craftsmen maintain their value. Therefore, to sustain technical excellence, older craftsmen must be enlisted as mentors to pass down their wisdom, insight, and experience to younger talent [13].

2.2. Quality in agile software development

Although software quality does not necessarily equate technical excellence, it is one of the outcomes of being technically excellent. Arcos-Medina and Mauricio conducted a systematic review of the agile literature on quality. They identified a catalog of factors, agile practices, and metrics that influence quality in agile methods [22]. Five critical success factors were proposed: teamwork practices, engineering practices, management practices, documentation practices, and testing practices. However, in most studies identified in this review, quality was not the primary topic of investigation. This literature study shows that more empirical work is needed to investigate perceptions of agile quality-related principles and practices and their correlation with achieving quality.

Timperi [20] discusses several engineering practices used by agile teams to assure quality, such as inspections, pair programming, test-driven development, coding standards, collective code ownership, and refactoring [20]. The study concludes that agile methods pay particular attention to validation while de-emphasizing verification. The author argues that this is problematic and recommends several quality assurance practices dedicated to verification and validation. Dingsøyr et al. [8] investigated the guidelines, principles, conventions, and other aspects concerning code quality that are known by developers in agile software development. They found that code comprehensibility and readability were often named first and most often by developers. Other mechanisms for code quality included structured naming, and detecting code smells (e.g., duplication, wrong abstraction, wrong naming, missing tests, side effects, a high number of parameters, noise, and cycles) [8].

Prechelt et al. propose "quality experience" [23], which is a quality assurance and deployment "mode" adopted by agile teams without the need for dedicated testers. This practice is characterized by three traits: (1) the team "feels fully responsible for the quality of their software"; (2) the team "receives feedback about this quality, in particular the

quality of their recent changes, that is fast (available early), direct (not be intermediated), realistic (coming from non-artificial settings); and (3) rapidly repairs deficiencies when they occur" [23]. This work shows that quality in agile teams is assured by a combination of technical and non-technical factors [23].

While the interest of researchers is expected to remain focused on quality as it is a highly sought outcome by the adoption of agile methods, an intriguing question is how quality differs from technical excellence. As a matter of fact, both encapsulate a set of practices and enablers, but software quality is steered more towards an outcome (e.g., free of bugs, maintainability, etc.) and technical excellence as our findings show is an outcome (e.g., sustainable code and a long term investment in technical decisions) and also a mindset. We contend that technical excellence contributes to quality, but has far reaching objectives. It intends to achieve and maintain long lasting technical qualities by promoting a growth and continuous improvement mindset and sustaining continuous learning.

2.3. The need to define "technical excellence" in agile

Tripp and Armstrong [24] identified three factors of motivation for adopting agile. These were the motivation to improve software quality, the motivation to improve efficiency, and the motivation to improve effectiveness. The quality factor included the motives of enhancing quality, improving engineering disciplines, and enhancing maintainability. The efficiency factor included motives of increasing productivity, accelerating time to market, and reducing costs. The effectiveness factor included the motives of enhancing the ability to manage changing priorities and improving alignment between IT and business objectives [24]. If the growing interest in agile is partly driven by achieving quality, then how does the agile manifesto advocate for quality and excellence? In addition, recent reviews of the literature [25,26] show limited interest on the topic of quality in agile. Since agile influences the value system of software teams, there is a merit to evaluate how agile teams achieve quality and technical excellence. Agile advocates for establishing values, and norms in the development team. However, we still know little how the agile value system enables teams to deliver better software.

2.4. Software architecture and process improvement

Other streams of research related to this study are software architecture and process improvement and learning. Research on agile software architecture has suggested a number of lightweight formal techniques [27] compatible with agile development such as refactoring [28], including architects in agile teams [29], architecture backlogs, and runway teams (i.e., teams that only work technical backlog items as opposed to user stories) [30]. The latter two practices are responses to the often observed challenge that customers prioritize work on functional requirements (i.e., user stories) over work on technical requirements [30]. Still, it is unclear how this body of knowledge links to the notion of technical excellence in agile software development.

Earlier software engineering research has focused on software process improvement initiatives that used normative maturity models, such as the Capability Maturity Model Integrated. Key findings of this line of research are that software process improvement initiatives can enhance software quality [31], that the improvement can be partially explained by learning [31], and that the critical success factors for such initiative include sufficient time and resources, organizational support, employee empowerment, and training [16]. Recently, research has shifted attention to process improvement in agile software development. Findings of this stream include that improvement actions relate to team-level learning [32] and that engineers are satisfied with agile improvement processes because of the short cycle time, which allows quickly observing improved outcomes [33].

The available literature, thus far, is fragmented and aimed at investigating a particular aspect of technical excellence (e.g., software architecture, software quality, etc.) in isolation. We still need to understand technical excellence holistically and especially the interplay amongst the various constituents shaping it. In addition, historically, software structure and other related technical artifacts have been studied in silos, assuming that the team and its social setting have little influence on how the software structure comes about. Conversely, our work examines technical excellence from an engineering, a social, and an organizational perspective. While we do not claim that this work answers all questions, it is a modest step towards a better understanding of technical excellence from an agile perspective in its broader engineering, social, and organizational contexts.

3. Methods

Our methodological decisions are based on pragmatism as an epistemological stance. Pragmatism advocates that research should focus on "practical understandings" of real-world problems, and the generated knowledge should have impactful implications on practice [34]. Pragmatism calls for selecting methods based on relevance to the problem being studied rather than the researcher's philosophical orientation [34]. We opted for a qualitative inquiry because we sought depth and breadth to further our understanding of the topic. In addition, we aimed to understand how technical excellence is achieved in practice to draw conclusions from practitioners' experiences.

Our qualitative inquiry focused on capturing agile practitioners' experiences. These experiences provided a depth of understanding about technical excellence in agile methods, which allowed deriving recommendations for implementing and fostering technical excellence. It is scientifically accepted and acknowledged that experience is a necessary and sufficient piece of knowledge in sciences [35]. Practitioners' experience is relevant to investigate our research questions. We explicitly looked for participants who took actions to implement technical excellence in their respective teams, which permitted us to understand the world as others experience it.

To permit a degree of flexibility in our conversations, we employed semi-structured interviews. Our interview questions (Table 1) can be sorted into three categories: introductory questions, which eased interviewer and interviewee alike into the discussion; core questions, which focused on the main topic; and probing questions, which followed up on specific details.

Subject selection

We interviewed 20 agile practitioners who we recruited from LinkedIn, and whose software development experience ranged from six to thirty-one years. To construct our sample, we used the profile search feature using the term "agile" that yielded over five million profiles. From this list, we randomly selected profiles without setting parameters and then we verified the person's suitability to participate in the study using the participant's profile description, especially their job descriptions and titles. We aimed for participants with long experience in agile software development projects, with a background in software development, and with interest and experience in implementing technical excellence. We identified those subjects as eligible participants that had a minimum of five years of experience in agile software development, had started their careers as software developers, and actively participated in implementing and fostering technical excellence in their teams. We examined a large number of profiles. Ultimately, fifty qualified individuals were invited to participate in the study, twenty of whom accepted our invitation. Table 2 summarizes the demographics of the participants, wherein "Exp. in Sw. Industry" indicates the number of years the participant spent working in the software industry and "Exp. as Software Developer" shows the software development experience of the participants. Affiliation is the company each participant was working for at the time of the interview.

Table 1
Key parts of the interview questions.

Introductory Questions
Can you please introduce yourself and talk about your experience? How do you define agile? What do you think of agile?
Core Questions
What does this statement from the Manifesto mean to you: "Continuous attention to technical excellence and good design enhances agility"? What is "continuous attention to technical excellence"? How does "good design [enhance] agility"? How do you foster "technical excellence" in an agile environment?
Probing Questions
Can you share with me examples of how technical excellence is implemented and fostered from your experience? Is there anything else you would like to add on the topic of "Technical Excellence" in agile?

Table 2
The study interviewees. * indicates interviewees who participated in the 1st focus group and ** in the 2nd member validation session.

#	Role	Experience		Affiliation		Country
		Exp. in Sw. Industry	Exp. as Software Developer	Business Sector	No. Emp.	
P1*	Sr. Agile Product Manager	20	12	Information Technology & Services	680	Germany
P2	Agile Coach	20	14	Information Technology & Services	2,369	Australia
P3	Agile Coach	20	12	Telecommunication	26,843	UK
P4*	Agile Delivery Specialist	12	8	Telecommunication Services	11,133	USA
P5	Scrum Master	14	10	Transportation	5,677	India
P6	Scrum Master/Team Agilist	15	10	Financial Services	768	USA
P7	Agile Coach	18	12	Information Technology & Services	40,993	Germany
P8	Project Manager	6	4	Finance Services	1,087	Spain
P9*	Project Manager	28	12	Automotive Manufacturer	4,654	Italy
P10*	Portfolio Manager	20	14	Information Technology & Services	3,059	USA
P11**	Program Manager	21	13	Information Technology & Services	345	India
P12	Scrum Master	31	16	Professional Training & Coaching	88	UK
P13	Senior Product Manager	18	14	Education	12,344	UK
P14	Project Manager	15	10	Information Technology & Services	99,353	UK
P15	Head Of Quality Assurance	4	7	Technology Startup	55	India
P16**	Product Owner	11	7	Information Technology & Services	309,284	India
P17	Lead QA Engineer	10	6	Education Management	312	USA
P18	Project Manager	8	5	Technology Startup	32	Germany
P19	Agile Coach	7	5	Travel & Tourism	268	Australia
P20	Scrum Master	7	5	Information Technology & Services	253	USA

"Business sector" was the industry sector of the employer and "No. Emp." its number of employees. This information was sourced from the companies' websites. The country is where the participant's company is registered. We ended up with a sample where all participants used and had experiences with Scrum. This was not intended, but it shows the popularity of Scrum in the industry.

Data collection

As the interviewees were widely distributed geographically, all interviews were conducted using Zoom, an audio–video conferencing tool. The interviews lasted 40–60 min on average and generated an average of 11 pages of text each when transcribed verbatim. The interviews were conducted by the first author in the period between March and August 2020. We used "Temi", an online transcription tool, to transcribe the interviews. The first author checked and corrected all transcripts where necessary and sent the transcripts to the participants for review (see section Section 5.4 for further details).

Data analysis

We used thematic coding to analyze the data, following the guidelines by Cruzes and Dybå [36]. Our analysis approach was inductive. That is, our interpretation of the data was not based on existing theory but rather based on the meaning that emerged from the participants' accounts. The iterative analysis began in the early stages of the data collection and continued throughout the study. The interview transcripts were coded by examining the data line-by-line through the lens of our research questions. Once the responses were coded, patterns were identified, suggesting a specific theme, a concept that organizes a group of repeating ideas related to the research question. After identifying and giving names to the basic meaning units, we grouped them into categories by similarity. We initiated the coding process and analysis as soon as interview transcripts became available so that we could monitor for saturation. We reached saturation [37] of some themes after twelve interviews (e.g., continuous improvement) and for other themes such as knowledge sharing and psychological safety at 20 interviews. For RQ1 & 2, the coding was initially performed by the first and third authors and for RQ3 by the first and second authors. We used peer debriefing sessions to critically review the codes and their categorization. During these sessions, the authors provided feedback on each other's coding decisions. This was done iteratively until a consensus was reached on a final list of codes and themes. We conducted an additional iteration of analysis (by the first and second authors) for this extended version of the paper to fine-tune the findings and identify relationships among categories. We decided to change the presentation and the configuration of the themes compared to the earlier version to improve integration. In this iteration, we used the qualitative data analysis tool NVivo (version 12) to identify relationships in statements that expressed claims between categories. We coded these claims as NVivo relationship nodes (i.e., nodes that express a relationship between two categories). We also used NVivo matrix queries, which helped visualize data segments in which informants talk about two categories at the same time. We examined these data segments for claims about causal relationships. When two categories had similar relationships to other categories, we aggregated these two categories to a higher-order category (e.g., feedback processes and knowledge sharing to continuous learning practices). These activities resulted in the model shown in Fig. 1.

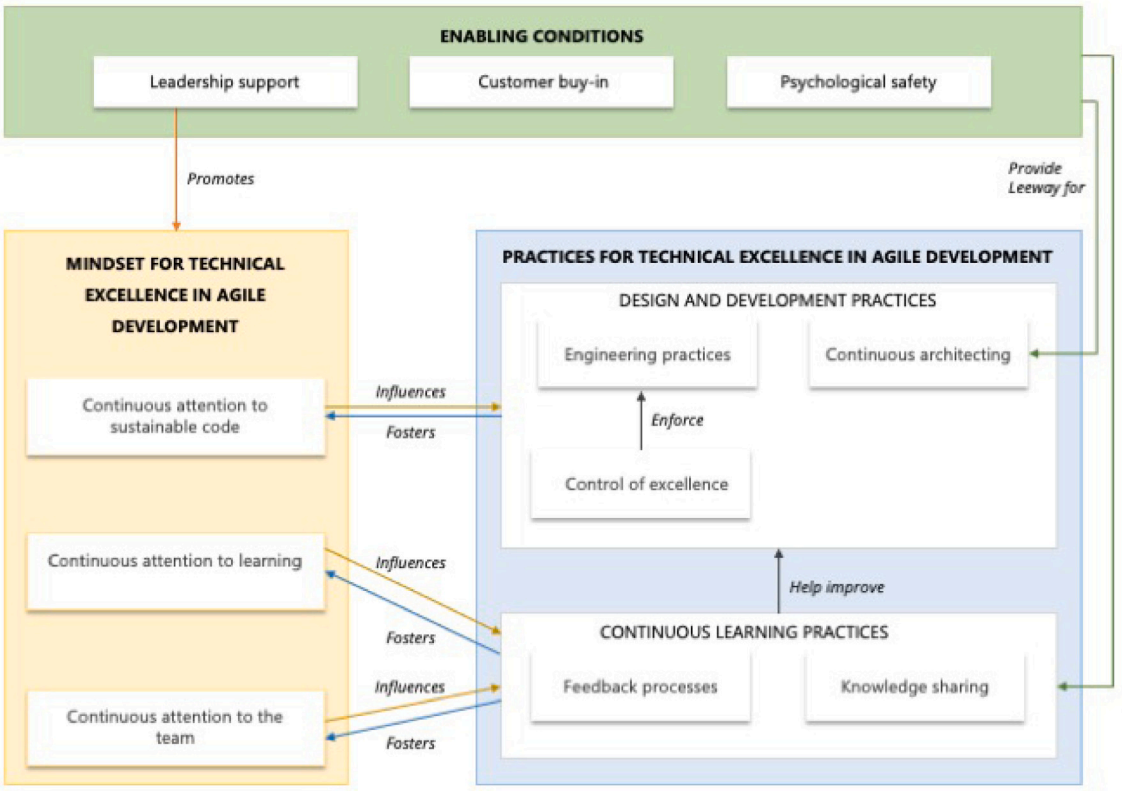


Fig. 1. Emergent model — Technical excellence in agile software development teams.

Member validation

We opted to use validation sessions for member checking, a technique for validating the emerging results. During these sessions, we presented our findings to the participants to check for accuracy and resonance with their experiences. While we invited all participants to this validation exercise, six participants accepted the invite. Due to their availability, we scheduled two separate sessions (see section ?? for further details).

4. Findings

Fig. 1 provides an overview of our findings. Our informants expressed that technical excellence has its roots in a mindset that embraces continuous attention to sustainable code, to learning, and to teamwork (the yellow box in Fig. 1). Our analysis shows that this mindset influences practices for technical excellence, which include design and development practices and continuous learning practices (the blue box in Fig. 1), which may, in turn, further foster this mindset. Our data further suggest that leadership support helps promote the mindset and that leadership support, customer buy-in, and psychological safety provide leeway for the team to practice continuous architecting and continuous learning (green box in Fig. 1).

4.1. Interpretation of Principle 9 (RQ1)

In response to **RQ1**, our participants emphasized that technical excellence in agile development results from a mindset, i.e., a collection of shared values and beliefs. For example, P3 stated: *"Technical excellence has a behavioral aspect to it. It needs the right mindset and culture."* In a similar vein, P10 emphasized: *"Technical excellence is first a mindset."* Our participants interpreted this mindset as consisting of three key elements: (1) continuous attention to sustainable code, (2) continuous attention to learning, and (3) continuous attention to the team. Table 3 shows these categories, their underlying codes, the number of occurrences of these codes in the data, and example quotes.

4.1.1. Continuous attention to sustainable code

Our informants expressed that their design and development practices for technical excellence are underpinned by a mindset that emphasizes continuous attention to sustainable code, i.e., software that is designed to be easily adaptable to future changes [38]. This mindset is constituted by four properties: (1) Long-term orientation, (2) commitment to clean code, (3) continuous attention to detail, and (4) craftsmanship.

Long-term orientation. Many of our participants were guided by a long-term orientation in their design and development work. For P1, the agile way of achieving high-quality software is *"to build software that has high quality, so it can be changed in the future in a sustainable way"*. In this perspective, an important goal is to design software such that developers can deliver modifications to the customer quickly with fewer bugs and at a lower cost of ownership, resulting in increased business agility. P20 elaborated on this idea: *"Technical excellence and good design is designing your product in a way that you do not commit to a single fact you know, designing adaptable products"*. For P6, designing with a long-term orientation means *"thinking in Lego bricks"* such that bricks can be easily exchanged and new bricks added, *"and nothing breaks"*. Such a future-proof code *'enhances the agility of your customers' business"* (P6).

Commitment to clean code. Many informants mentioned that a long-term orientation led them to follow the principle of "clean code", denoting code that can be easily understood and changed. P9 explained that *"the key to writing good clean code is to know how to avoid complexity"*. P19 asserted that complex code hinders scalability, stating: *"Complexity is the foe of scalable, robust and reliable software. Technical excellence also means that developers need to ensure writing clean code and refrain from writing complex code"*. According to our informants, clean code is code that can be understood even by readers that are unfamiliar with the code. Our participants expressed that a key strategy

Table 3
Mindset for technical excellence (RQ1) - Categories, codes, number of occurrences in the data (N) & examples quotes. Hyphen in the second column implies that the category is abstract, i.e., the code constitutes a category in its own.

Category	Code	Example quote
Continuous attention to sustainable code	Long-term orientation	*"[Technical excellence means to me] the focus on what you need to achieve in the longer term. It's about sustainability, quality. So, it's more about doing something which can be reused, doing something that can be like creating a template for others to follow" (P11).*
	Commitment to clean code	*"[Technical excellence] also means simplicity. Simplicity in writing code, simplicity in the design. Complexity is a foe of scalable, robust and reliable software. Technical excellence also means that developers need to ensure writing clean code and refrain from writing complex code" (P19).*
	Continuous attention to detail	*"[I]f you see something that doesn't really conform to [technical excellence], you need to kind of nip it in the bud really quickly ... That could be an attitudinal piece" (P2).*
	Craftsmanship	*"Technical excellence is also achieved by craftsmanship and creativity"* (P4).
Continuous attention to learning	Growth mindset	*"Not every person has the technical excellence in one particular point of time, one particular moment. Because technical excellence is about experience, learning, fail[ure] and so ... You need to study, you need to learn, and you need to improve and also to work for experience in this field. And sometimes you fail in this job during this journey ... that's okay" (P8).*
	Continuous improvement mindset	*"Agile advocates experimenting, observ[ing] the outcome and then we build a learning out of it, then it becomes a belief. These beliefs and practices then become engrained in our habits. That's how we develop a mindset" (P5).*
Continuous attention to the team	–	*"For me, technical excellence is about egoless development and uncovering better ways of developing software by doing it and helping others do it. So, at the end excellence is the excellence of the team" (P13).*

for ensuring understandable code is simplicity, or as P20 put it: *"Simple code allows maintaining, debugging, refactoring, and adding features with limited knowledge of how the entire system works"* (P20). P4 emphasized that code that is easily understood is also easily changed, which ensures sustainable software: *"Agile is about delivering efficient code, easy to change for future requirements. This is agility"* (P4).

Continuous attention to detail. Our informants emphasized that sustainable code not only requires a long-term orientation and clean code as guiding principles when making important decisions, it also entails continuous attention to detail from the start to the end of a project. For P13, technical excellence is *"just a software without any kinds of bugs and good design ... from the beginning what you start building the product to the end"* (P13). P7 even said that *"it should be a mindset thing just to leave code better than you find it"*. P1 expressed that technical excellence means a mindset of building high-quality code from the start, even under tight deadlines: *"And this means that even in a very short implementation cycle, you should pay attention to quality and to technical excellence, which means you should not allow to implement intermediate solutions in the hacky way in ... and hope that later you will have to have time to fix them"* (P1). She prefers a rather modest scope for each sprint such that *"the features are implemented in quality. And then you implement simple features, but always in high-quality, with continuous attention to technical excellence"* (P1).

Craftsmanship. Many informants expressed that a mindset of continuous attention to sustainability is rooted in their craftsmanship, i.e., the view that software development is skillful work and that software developers take pride in the outcomes of this skillful work [13]. For P4, *sustainable software is achieved by craftsmanship"* (P4). P3 described agile enables craftsmanship: *"I have seen it [agile] fostering the craftsmanship element in the team. ... It takes away the process of treating the development resources as, for example, factory workers in the manufacturing sector have been evolving"*. He further explained that craftsmanship is what leads developers to take care of quality: *What craftsmanship is about is this feeling the pride of what you are doing, perceiving, and acknowledging that I am contributing something for some good and my work is appreciated ... when you have that mindset in your developer, he's going to take care of quality himself..."*

> **Continuous attention to sustainable code:** For our informants, technical excellence had its foundation in a mindset supported by technical procedures. The essence of this mindset is a feeling of pride about skillfully producing sustainable software. This mindset leads people to continuously focus on clean code that is easy to scale and adapt for future business needs.

4.1.2. Continuous attention to learning

While continuous attention to sustainable software involves a long-term orientation regarding the technical products of design and development work, our informants also emphasized a long-term orientation regarding the knowledge and skills that enable software development. We term this mindset continuous attention to learning. It is constituted by two properties that operate at different levels: (1) a growth mindset related to each developer's individual skills and (2) a continuous improvement mindset related to a team's practices.

Growth mindset. In several statements, our participants highlighted the role of a growth mindset, i.e., the belief that people can develop their skills through dedication and hard work [39]. For example, P10

emphasized the importance of being open to new learning opportunities: *"That's the first thing you need to look for. A mindset attitude. It's soft. So, they are personal traits, openness, flexibility"*. According to our informants, a growth mindset entails a belief in their own and others' abilities to learn from failures. For instance, P12 acknowledged that technical excellence *"also requires some trust in yourself maybe and in others, . . . and somehow a willingness to fail, because this approach is really trial and error"*. P5 tells the story of a developer that lacked a growth mindset and left the team: *"We had a team member who ... was inflexible, close-minded and unwilling to learn... Eventually, he voluntarily left... [T]his example shows that if you can't experiment, self-reflect as an individual and a team, and adapt, you do not have the required mindset. This ... particular individual did not have a growth mindset"*.

Continuous improvement mindset. Many statements indicated a mindset that appreciates learning not only at the level of a developer's individual skills but also at the level of a team's collective practices. For instance, for P4, *"[a]gile is also about continuous learning and improvement. Each iteration is a learning opportunity. We learn, and then we take actions to improve" (P4)*. P16 referred to this mindset as a culture: *"technical excellence evolves around the culture in which continuous improvement is key"* (P16). P12 shared the impression that a continuous improvement mindset is lacking in many organizations despite the use of agile methods: *"Scrum is based on continuous improvement. So, if we don't learn from our approaches, we can't improve ... And unfortunately, this technical excellence, . . . in most companies, . . . is ... really ... maybe a taboo topic, they are trying to do it in a way, but they are not really succeeding"* (P12).

4.1.3. Continuous attention to the team

For our informants, a mindset for technical excellence implies attention not only to code and learning but also to the team. For P15, technical excellence refers to *"culture in the sense that it is not a one-man army approach"* (P15). As P4 highlights, a team mindset is critical for teamwork and learning: *"So many times ... I have seen ... an expert developer that ... doesn't want to spread their knowledge, or they want to work with themselves. ... They want just to have user stories and then tie up the code into the environment. It is more, to be technical excellence, we have to have a small community within the company ... it's about how you collaborate with other members of the team developers, designers, the business people, and how you share your knowledge and help others do the same"* (P4). In a similar vein, P2 told the story of a developer that acted like *"a lone wolf"*, disregarded other developers' feedback, thus contributed code that violated clean code principles, and caused "lots of problems because suddenly a number of things started breaking". (P2)

Continuous attention to learning and to the team: Our participants highlighted that technical excellence involves more than developers' commitment to sustainable code. Technical excellence also originates from a mindset that values teamwork and continuous learning at individual- and team level.

4.2. Practices for fostering technical excellence

Recall that, for **RQ2**, we asked how agile teams foster technical excellence. In response to this question, our participants reported that technical excellence is nurtured by particular practices, i.e., by patterns of recurrent actions in software teams. These practices were design and development practices (engineering practices, control of excellence, continuous architecting) and by continuous learning practices (feedback processes and knowledge sharing), which help improve the design and development practices. By particular practices, i.e., by patterns of recurrent actions in software teams. These practices were

4.2.1. Design and development practices

Even though technical excellence means more to our informants than attention to technical decisions, concerns, and practices, they emphasized that technical excellence is not possible without effective design and development practices, including engineering practices, control of excellence-practices, and continuous architecting.

Engineering practices. For many of our participants, key ingredients for ensuring technical excellence are engineering practices that help teams develop high-quality and sustainable code. The practices mentioned by our informants include automated testing, automatic code analysis, code review, documentation, peer programming, test-driven development, and unit testing. For instance, P3 said: *"I would immediately relate technical excellence in the ways you build up the software, the tools, and the practices that are used to deliver the technical aspect of the product, the actual software code and the design"* (P3). P11 highlighted that automation plays a key role in engineering practices that produce sustainable code: *"So, engineering excellence, whatever you do in terms of engineering ... So, for example, if I'm taking my shipment in a build, it's going to take three weeks, for example, and with X number of features, try to optimize it, try to ... automate stuff rather than going manual ... Optimizing, automating, focus on quality"*.

Control of excellence. Our participants mentioned several techniques for controlling the goals of technical excellence and enforcing those engineering practices that contribute to technical excellence. Practices for control of excellence include a Definition of Done [40] (i.e., a set of criteria to determine if a deliverable is complete), enforcing compliance with coding results, enforcing prescribed processes, and measurement. For example, P11 explained: *"Technical excellence is also controlled by the "Definition of Done", which is a set of quality checks to make sure code and other artifacts meet our requirements for quality"*. P4 explained that control of excellence does not supersede the people factors: *"We control technical excellence via the Definition of Done, but as I mentioned before, we need to invest in people. We need to coach them to be technically excellent"* (P11).

Continuous architecting. Our interviewees emphasized that building sustainable software required taking regular breaks from developing new functionality and turning attention to the software's architecture. We refer to these practices as continuous architecting practices because they describe an engagement with architecture throughout the project [41]. Notwithstanding the often stated emergent nature of architecture in agile software development [4], many participants highlighted the need for deliberate planning at a project's outset. For example, P19 said that *"you need to have a stable ground to build"* (P19). In line with the quest for sustainable software, P4 recommended that initial architecting should involve looking into the future: *"First envision the ... full product. Like ... if we go down the line two years from now, how would the product look like and translate this also in the technical part. Just leave some room [for] ... some feature that may be implemented six months from now, one year from now"*. While initial architecting is important for assuring sustainable software, this initial architecture needs to be revised throughout the project, according to our informants. As P16 explained: *"Once you've got a solution to high-level design and you start your development ... every interval you should always ... revisit the solution that you have initially designed"*. (P16). P9 pointed to the limits of initial architecting and said refactoring is often needed when reconsidering the architecture: *"I haven't come to the project where everything can be anticipated in the initial design so that there is no need to refactor"* (P9). Given that architecting does not produce new functionality, our participants emphasized that it was important to give the teams time for continuous architecting, for instance, by defining technical backlog items, which, as P7 put it, help ensure that *"resources ... are not [fully] ... committed to functionality .... They take maybe eighty percent of capacity... they have some time just to solve the technical things. So, they have a technical backlog'* (P7).

Table 4
Practices for technical excellence (RQ2) - Categories, codes, number of occurrences in the data (N) & example quotes.

Category	Code	Sub-code examples	Example quote
Design and development practices	Engineering practices	Automated testing, automatic code analysis, code review, coding rules, documenting, peer programming, test-driven development, testing by quality assurance staff, unit testing	*"It's really about technical best practice ... If you're going to have to respond to change quickly, ... you want to be developing things that are ... as robust as possible. And the way to do that is to use best ... engineering practice ... like design patterns, ... automated testing, test-driven development, and all those ... things ..." (P2).*
	Control of excellence	Definition of Done, enforcing compliance with coding rules, enforcing processes, measurement	*"Technical excellence is also controlled by the "Definition of Done", which is a set of quality checks to make sure code and other artifacts meet our requirements for quality." (P11)*
	Continuous architecting	Architecting from project start, giving people time to architect, including architects in the team, looking ahead, refactoring, technical backlog	*"So using the right solution, which can meet all your requirements and ... keep on revisiting your solution architecture at different levels of your development so that you can identify if there is any gap or if there is any requirements which is not yet met." (P16)*
Continuous learning practices	Feedback processes	Continuous development, early prototypes, experimenting, improvement routines, iterative development	*"[Y]ou need to have the retrospective or a moment where you acknowledge these things and take some actions... like next time we need to do it different, we need to try this, we need to try that until we reach an optimal level of technical excellence..." (P20)*
	Knowledge sharing	Asking for help, common code review, communities of practices, education and training, helping each other, job rotation, knowledge sharing sessions, mentoring and coaching, self-development	*"Another example is when a new team member join[s], we check the quality of their code. If it is not within our expectations, then he is assigned ... a mentor for a period of time until he learns the craft of writing good software code. ... [I]t is a loss of a resource for some time, but it is worth the investment and the long term reward." (P12)*

Design and Development Practices: Engineering practices (e.g., automated testing, coding standards) are a cornerstone of technical excellence according to our participants. Control-of-excellence practices such as Definition of Done help enforce the disciplined use of these practices. Sustainable software also requires continuous architecting, implying that engineers envision the future of the software to define an initial architecture and have the capacity throughout the project to revisit the architecture and refactor the code as necessary.

4.2.2. Continuous learning practices

All our participants expressed that technical excellence is achieved not only through design and development practices but also through continuous learning practices, i.e., through recurrent activities that continuously enhance the team members' individual knowledge and the team's collective knowledge that materializes in the design and development practices. In our analysis, we identified two broad types of continuous learning practices: feedback processes and knowledge sharing.

Feedback processes. Feedback processes are recurrent activities through which teams obtain, share, and reflect upon information about the outcomes of their actions [42]. Among the most frequently mentioned feedback processes were improvement routines and continuous development. Improvement routines, such as retrospectives in Scrum, are ceremonies in which a team looks back at their recent activities and identifies actions for improving these activities in the future [32]. Many informants shared that such routines helped improve their design and development practices. For example, P15 told how he used internal meetings to improve engineering practices, specifically the use of automated testing: *"this was for one client where everything was being done manual... We discuss ... that you are doing the same things again and again because if they have frequent deliverables, you are ... executing the same [tests] ... again and again. ... the team was not knowing the automation techniques ... So, we started with ... internal meetings where we used to ... go and learn and implement a little bit. So, by the end of five, six sprints, we were able to automate a few of the pages"* (P15). The second frequently mentioned feedback process was continuous development, i.e., integrating, deploying, and testing code at a high frequency, such as several times a day [43]. P1 experienced that progressing towards continuous development helps teams to recognize potential for improvement: *"[You] should strive for ... a very fast release cycle ... Let's say you start with a two-week release cycle, then ... release after each week ... [t]hen later go to daily deployments and then later deploy whenever a feature is ready. And by doing this, you will uncover the problems that a team has with technical excellence. Maybe they don't have a good test coverage, maybe they have some components which have a lot of interdependencies and cause a lot of side effects... [S]hortening the release cycle ... will uncover problems with missing technical excellence early and the team can work on it"* (P1).

Knowledge sharing. Our data abounded with statements that emphasize the key role of knowledge sharing to ensure that developers acquire and improve skills required for technical excellence. Knowledge sharing occurred and was promoted through many different mechanisms, such as engineers asking other team members for help, common code reviews (i.e., sessions in which the entire team examines and reflects upon a given code segment), engineers helping each other, and mentoring and coaching, and self-development (see Table 4 for more sub-codes). For example, P6 narrated how brown bag sessions (a type of knowledge sharing sessions) enabled developers to use start-of-the-art engineering practices: *"For example, we have these brown bag sessions when some people ... give these workshops about how can you use that library for achieving that or how in this organization we use that library or how ... these organizations use Jenkins to achieve that. It's important ... to keep the people continue learning, the people always improve their technical knowledge so we can start using all the tools that we already have on our hands and use it well"* (P6). P15 explained how mentoring helps ensure that skill gaps, and the resulting shortcomings in the code, are closed by *"making the more experienced person as a buddy of the person who is having less expertise. So, with the buddy, you can ask the person to mentor daily work deliverables. If there are any shortcomings, the reviews can be submitted by the buddy to be less expertise person"* (P15). Many interviewees also emphasized the key role of self-development (e.g., consuming e-learning material where others share their knowledge digitally).

Continuous learning practices: Teams that strive for technical excellence rely not only on specific design and development practices; they also leverage continuous learning practices. These teams make disciplined use of improvement routines (e.g., retrospectives) and of continuous development (e.g., continuous integration and delivery), and they create and maintain many different avenues for knowledge sharing.

The interplay between mindset and practices

As illustrated in Fig. 1, mindset (which focus on what people believe and value) and practices (which focus on what people do) influence each other according to our participants. A mindset of continuous attention to sustainable code influences design and development practices. For instance, P7 shared how values such as sustainable code influence design and development activities: *"We started from values … It's when you give such basics, they start to care about the product, and they start that from simple things. They started from unit tests. They started from integration tests. They jump to automation tests later on to some kind of deployment improvements and so on. So, it's just, I would say, start from values"* (P7). Design and development efforts, in turn, can foster a mindset of paying continuous attention to sustainable software. For example, P20 narrated how conversations during continuous architecting strengthen a mindset of continuous attention to sustainable software: *"[W]hen one developer comes with a solution like, I want to implement this, I always challenge them, like, why? What's the alternative? What's the plus and what's the minus? … ? Just how would this scale … ? That way I'm challenging them to think ahead … So, people are always ready and trying to find the best solution and find architecture and software, which is flexible and scalable"* (P20).

Our interviewees also shared how a mindset of continuous attention to learning and of continuous attention influences continuous learning practices, which can, in turn, strengthen a mindset of paying continuous attention to learning and to the team. As P5 put it: *"The mindset is at the core of everything we do and practices impact belief and vice-versa. Agile advocates experimenting, observe the outcome and then we build a learning out of it, then it becomes a belief. These beliefs and practices then become engrained in our habits"* (P5). P12 observed that a growth mindset informs the way and the intensity of knowledge sharing activities in a team, which, in turn, fosters a mindset of stronger attention to the team: *"So, it's again, the cultural thing I keep mentioning…[I]f all my teammates are people that are interested in self-development, then the whole thing will be or if there is just one person or two people in my team that don't know how to do that, they will absorb the energy somehow from the others. And they will learn from them, the benefits of self-development"* (P12).

4.3. Enablers of technical excellence

While **RQ2** deals with how agile software development teams develop and nurture technical excellence through their practices, **RQ3** focuses on enabling conditions for technical excellence. In this context, enabling conditions refer to properties of a software development team's organizational context that are conducive to technical excellence. Our analysis indicates three important enabling conditions: leadership support, customer buy-in, and psychological safety. Table 5 shows the number of occurrences and example quotes. Our data suggest that these conditions contribute to technical excellence in two major ways (see also Fig. 1). First, leadership support can promote a mindset for technical excellence. Second, all three enabling conditions provide leeway for continuous architecting and continuous learning practices.

Leadership support. From the perspective of our participants, leadership support plays two important roles. First, leaders can be role models that demonstrate a mindset for technical excellence, making it more likely that employees endorse the mindset. For example, P12 emphasized that leaders with a growth mindset can inspire employees to adopt a growth mindset: *"[E]verything starts from the team lead and the management upwards. If these people don't have the learning mentality or they are focused only on delivering the project, . . . [then] they lack … the people skills to teach the people that it's important to learn and to improve themselves… people look up to management … So, the team leader is seen as the high power and as the role model in the team"* (P12). Second, leaderships support is important because it provides leeway for the development team to engage in feedback processes and in architecting. For example, P10 shared how feedback processes depend on the tone set by the leader: *"[A]though some people might realize that the leader is not totally okay with their opinions, with his designs, they don't speak up. But when you have that leader that is flexible, open-minded, and horizontal …, then their team members tend to speak up their actual thoughts"* (P10).

Customer's buy-in. As with leadership support, our informants stressed that customer buy-in is important because it provides leeway for the development team to invest in technical excellence. P20, for example, explained the importance of educating the customer about the value of technical excellence, stating that *"there's always that convincing part, you know, you need to convince the customer to wait a bit more because we're applying some technical excellence issues here that will be of value in the future"*. P16 stressed the need for the development team and customer to be "on the same page": *"So basically, when all the team members are on board and on the same page, when you get a customer's buy-in as well, . . . you started … solutioning things, which is suited for that particular project or that particular solution, which actually helps … the customers to solve certain problems in their line of business. So, it helps you to grow. It helps you to design effective solutions"* (P16).

Psychological safety. Psychological safety is the freedom to speak one's mind, confident that there will not be negative consequences for making mistakes or taking initiatives [44]. Our participants expressed the belief that promoting a sense of psychological safety makes people comfortable about showing initiative (e.g., architecting initiatives), investing in self-development, and striving to excel. P19 explained that he and his team operate in a work environment where they *"discuss [their] failures, shortcomings, and imperfections, whether in technical process or individual behavior"*. P17 highlighted the importance of a work environment where people do not fear consequences from bringing up new ideas: *"How do you nurture technical excellence in Agile or Scrum? I would say the first thing is … respect each other's values. If someone comes with some new idea to you, do not suppress that. You should always encourage everyone else' new ideas"* (P17). Similarly, P1 drew a direct link between psychological safety and continuous learning: *"So, if you give the team the safety to know they have the time to do these kinds of quality improvements to enhance their technical quality, their technical excellence, the technical excellence will get better over time"*.

Enabling conditions: Leaders that act as role models are critical for promoting a mindset that endorses sustainable code, learning, and teamwork. Moreover, leadership, customer buy-in, and psychological safety provide leeway for software development teams to invest time in continuous architecting and learning, both of which are important for technical excellence.

5. Discussion

5.1. Mindset for technical excellence

As illustrated by Fig. 1, a key finding of our study is that efforts for technical excellence in agile software development are rooted in a mindset that emphasizes continuous attention to sustainable code, to learning, and to the team. This finding highlights that, for agile practitioners, technical excellence is about more than architectural concerns. While a long-term orientation towards code and its architecture is an important part of it (i.e., continuous attention to sustainable code), another important part of it lies in a long-term commitment to fostering knowledge and skills and in commitment to the team as the social

Table 5
Enabling conditions for technical excellence (RQ3) - Categories, number of occurrences in the data (N), examples quotes. There is no column dedicated to codes because these categories are abstract, i.e., they emerged in the first coding iteration (line-by-line) and formed categories in their own.

Category	Example quote
Leadership support	*"As I said, ... technical excellence is first, a mindset. Having a technical lead or a Scrum Master or a project manager, but a leader, having a leader that has this mindset of ambition, of excellence, having that leader is one first step because this person can be a role model for everybody else." (P10).*
Customer buy-in	*"[I]n this project ... I ... managed to convince the client that this will pay off later if we do it now. So, one of the teams was dedicated to the technical refactoring..." (P9).*
Psychological safety	*"I think the last topic on this would be the fact that we are still learning how to allow people to make mistakes. It is in the Agile mindset that people should embrace actually the possibility of failing and learn from that." (P14).*

unit in which much of the learning takes place that enables technical excellence.

These findings connect to but also extend software architecture research in several ways. The long-term orientation that lies at the heart of a mindset of continuous attention to sustainable code parallels the long-term orientation in software architecture, which is concerned with those *"design decisions that have a long-lasting impact"* [45] on software. Moreover, much as software architecture focuses on the non-functional qualities of software [46], our informants' interpretation of technical excellence focused on those technical properties of code that made code sustainable, as opposed to properties related to functional requirements. Indeed, the notion of sustainable code is closely related to the non-functional quality criterion of maintainability in software architecture research [46] and quality standards [47], although the agile practitioners in our sample were concerned with the ease of changing the code not only during maintenance but from the outset of the project. As many informants attested, this focus on sustainable code, as opposed to other non-functional quality criteria such as security or performance, is due to the key role of the ability to change code in agile software development [3,4]. Hence, perhaps not surprisingly, technical excellence means for agile practitioners a concern with those non-functional aspects of software that enable timely and efficient adaptation to changing business needs. As such, one contribution of our study lies in uncovering what agile practitioners mean from a technical point of view when they use the term technical excellence.

While software architecture research helps put our findings around a sustainable-code mindset into context, there is an important difference between our findings and the processes that are typically advocated by software architecture research. A substantial part of software architecture research, including the more recent research on architecture in agile software development [48,49], focuses on formal processes through which dedicated architects ensure the developers' attention is drawn to architectural issues [15]. Notwithstanding the value of such processes and of the inclusion of dedicated architects in many settings, our category of a mindset of continuous attention to sustainable code shows that a substantial part of the architecting work in agile software development teams has its origins not in formal processes led by dedicated architects but in the developers' commitment to developing architecturally sound software and getting better in it. This empowered role of developers is in line with agile software development principles, among which are empowerment and self-organizing teams [6], and with the notion of craftsmanship, which emphasizes a genuine interest in and pride in skillful work among software developers [5,13]. Importantly, these findings do not imply that formal processes and dedicated architects are irrelevant in agile software development. Indeed, some of our informants shared that they sometimes include dedicated architects in agile teams, especially when complex architectural issues needed to be resolved. However, our findings show that a strategy of creating a commitment to sustainable software can be complementary to traditional software architecture approaches relying on dedicated architects and formal processes. Such a complementary approach of empowering developers to continuously take care of architecture could also help alleviate some chronic challenges of software architecture, including lack of authority, the ivory tower syndrome, and procrastination [50].

While our findings show that a mindset of continuous attention to sustainable code is an important pillar for technical excellence in agile teams, they also point to important roles of mindsets for continuous attention to learning and to the team. These mindsets foster the continuous learning activities that help improve design and development practices to ensure technical excellence. Although much work has looked at continuous learning processes in traditional [16,31,51] and recently also in agile process improvement initiatives [52,53], this work has rarely examined the role of mindsets, despite a call for research to capture the emergent, bottom-up facets of process improvement [54]. An important contribution of our research is thus to highlight the roles that a growth mindset, a continuous improvement mindset, and a team mindset play in fostering the continuous learning processes that contribute to technical excellence in agile software development.

5.2. Practices for technical excellence

Our findings on RQ2 suggest that design and development practices (including engineering practices, control of excellence, and continuous architecting) and continuous learning practices (feedback processes and knowledge sharing) are cornerstones of technical excellence. The importance of each of these practices is strongly established in software engineering research, though rarely in relation to the specific notion of technical excellence. Engineering practices and control of excellence practices are widely discussed in agile methods texts (e.g., [4]), in empirical software engineering research (e.g., [8,20,40,55]), and in movements such as DevOps [56]. The practices subsumed under continuous architecting (see the sub-codes in Table 4 are well documented in software architecture research [4,27,41,48,57,58]. The key role of feedback processes is well established both in continuous software development [4,23,42,59] and in process improvement [32,52]. The importance of knowledge sharing is also widely appreciated (e.g., [60]). Against this backdrop, our contribution is twofold. First, we establish a link between from these practices to technical excellence. Second, we document the links between these practices. Indeed, while software architecture research has focused on design and development practices and process improvement research has focused on continuous learning practices, neither of these two literature streams has paid strong attention to the interplay of these two types of practices, even though their interplay is critical for technical excellence according to our informants.

5.3. Enabling conditions

Our findings on RQ3 show that leadership support, customer buy-in, and psychological safety are important enabling conditions for technical excellence. While leadership support helps promote a mindset for technical excellence, all three enabling conditions provide leeway for activities beyond engineering practices and their control, namely continuous architecting and continuous learning processes. The role of leadership support in promoting mindsets or norms is widely researched [61]. The finding that leadership support, customer buy-in, and psychological safety provide leeway for continuous architecting and continuous learnings connects to findings both from software

architecture and process improvement research. Software architecture research, especially its agile stream, has emphasized the difficulties of prioritizing architecture work in agile settings where customers are often most interested in functionality [29,30], even though architecting efforts pay in the long run due to more sustainable software. Similarly, process improvement research has pointed to the problem that managers often prioritize short-term throughput over process improvement efforts, which pay only in the long term [62]. In line with this work, we find that providing leeway for architecting and improvement activities is important because it allows developers to devote sufficient to activities that are beneficial in the long run. Notwithstanding these commonalities to prior work, our study makes two contributions. First, our findings show that creating a mindset that appreciates sustainable code and learning can help solve the problem that short-term benefits (i.e., immediately available functionality) is often prioritized over work that yields long-term benefits (i.e., more sustainable software, greater knowledge). By creating mindsets that appreciate these long-term benefits, individuals and teams enact a more long-term orientation in their daily practice where they value architecting and learning as key activities for achieving technical excellence. Another contribution of our work concerns the influence of psychological safety on technical excellence. Even though software engineering research has shown that process improvement revolves strongly around team learning [32], and even though psychological safety is a key concept for team learning [44], psychological safety has received relatively little attention in software engineering research, particularly in relation to technical excellence and quality. Our study shows that psychological safety plays an important role for technical excellence because it promotes those activities (i.e., continuous architecting and learning) whose benefits will accrue only in the long-term and will, hence, be subject to uncertainty [62]. As our data shows, psychological safety helps make developers comfortable about engaging in such activities even if these activities mean delaying functionality that customers are pushing for at the expense of uncertain future benefits.

Our work does not only complement existing research but also brings a new breeze to this area of research. Some of our conclusions suggest that more work should be steered towards investigating non-engineering enablers, such as leadership support, psychological safety, and continuous attention to the team, and how they influence a software development team's ability to attain excellence. By shying away from these socially tuned enablers, we may be failing to acknowledge their power and subsequently impact the software engineering practice with non-technical recommendations.

5.4. Limitations and validity

The following methodological issues may impact the conclusions we draw from the data:

Scrum-focused data: All our participants practice Scrum, which was not intended, but illustrates the popularity of this agile implementation in the industry. Given this constraint in our sample, our findings may not be transferable to non-Scrum implementations. However, we believe this is a minor limitation. It is safe to claim that agile methods (e.g., Scrum and Crystal) share similar value system.

No observation data: The research questions two and three were investigated using practitioners experiences, as captured in the interviews, and not direct observations of agile software development teams. Direct observations may yield additional findings. What people actually do in practice may further advance our understanding of the investigated phenomenon.

Interviewee Transcript Review: This happened when the interviews transcripts became available [63]. We asked our participants to review the transcripts of their interviews. Eighteen respondents indicated that their transcripts were accurate, reflecting their responses; two did not respond.

Member validation. We opted for member validation of our findings. Member validation involves sharing research findings with the participants at the end of the study, and it is intended as a verification procedure to enhance the study's credibility [64]. We invited all our interviewees to participate in member validation sessions. However, only six participated in this exercise out of the twenty invited. We arranged two member validation sessions, one with four and the other with two participants. We ended up with this configuration due to the participants availability. Participants were presented with our interpretations of the data and invited to comment on the findings. The participants of these member validation sessions were given the opportunity to either confirm or deny that the summaries of findings reflect their views, feelings and experiences. The member validation sessions for this study were constructive and very supportive of the findings. We made minor revisions according to the feedback we received in the two sessions, but there were no major changes to the findings. We made the two member validation sessions transcripts available here.[2]

Saturation. A common standard for conducting qualitative research [37] is saturation, which involves adding more participants to the sample until reaching a point where no new additional codes, themes, or information emerges. We reached saturation when new data analysis became redundant with themes already identified. We initiated the analysis of the data early in the process in parallel with data collection and continuously monitored for saturation of our codes and themes. We did not see the need for further interviews after 12 participants for RQ1 & RQ2, as every code was sufficiently explained in depth by the data, and there was enough data to answer the research question. Saunders et al. explain that saturation is reached when "additional data do not lead to any new emergent themes" [37]. For RQ3, this requirement was met at the 20th interview. We adopted only codes we deemed saturated by the data of the available 20 interviews.

Verifiability. To allow the verifiability of our data, we made it available here[3]; to preserve the anonymity of our participants, we anonymized the interview transcripts.

6. Implications

6.1. Implication on practice

In this section, based on the experience of practitioners in our sample and the discussion presented in Section 5, we propose potential recommendations for the implementation of agile software development methods. Table 6 documents the implications we draw from the data. The level is the owner of the implication of a particular recommendation, i.e., the party responsible for implementing the corresponding actions. The recommendations are proposed measures to be implemented to enable a software development environment where technical excellence is attainable. These recommendations are anchored in our data. Hence, the last column establishes the links to the sources of the recommendations in the practitioners' accounts.

Organization

The ownership of having a shared goal for technical excellence, creating a psychologically safe working environment, and supporting the development team in its quest for technical excellence rests at the organizational level. Framing the goals for technical excellence does not only bring transparency to the conversations but also creates accountability. Once the goals are shared, a contract is established between the organization and the team, which makes the team accountable. For instance, if one of the goals is to reduce the number of defects per Sprint release as a target, it goes without saying that the customer will check the attainment of this expectation in the next Sprint. This aspect

[2] https://figshare.com/s/0f30e59c85c2b145e019.
[3] https://figshare.com/s/5798cc8e800a00429c0f.

Table 6
Recommendations on the Implementation of Technical Excellence in Agile Software Development.

RQs	Level	Recommendations	Participants
RQ1	Organization	Implement apprenticeship program to promote craftsmanship	P3 & P4
		Establish a program to promote peer-to-peer learning	P6 & P15
	Team	Implement a knowledge base for learning the principles of clean code	P1, P4, P9 & P20
		Establish standards to follow clean code principles	P1, P6 & P20
	Individual	Remain curious about innovative and contemporary practices	P5, P12 & P10
		Engage in self-learning to grow your technical knowledge	P5, P12 & P10
		Remain open to self-improving	P4, P12 & P16
RQ2	Organization	Encourage the team to experiment, learn and innovate	P1, P13 & P15
	Team	Develop explicit, robust, and sustainable criteria for DoD	P4 & P6
		Revise software architecture decisions iteratively	P4, P7, P16 & P19
	Individual	Remain consistent with standards established by the team	P1, P13 & P15
RQ3	Organization	Create a shared goal for technical excellence	P1, P9, P14-15 & P18
		Create and promote a psychologically safe environment	P1, P4-5, P7, P10, P13-14, P17 & P19
		Provide support to the team for the pursuit of technical excellence	P4-5, P7, P11, P14, P16, P18 & P20
		Invest in ongoing training of the development team	P3-4, P6-9, P12-13, P15 & P17-18
	Team	Share and promote knowledge	P1-4, P6, P8-9, P12-15, P17-18 & P20
		Collaborate and exercise egolessness	P8-9, P13, P15, P18 & P20
		Support each other in the pursuit of technical excellence	P8, P12-13, P15 & P17-18
		Educate the customer in the value of technical excellence	P8 & P20
	Individual	Remain intellectually curious	P3, P12, P15 & P17
		Remain motivated	P9 & P13-14
		Invest in self-development	P3, P12 & P17

of accountability helps to ensure that the team adheres to the goals. So, the moment the goals become known, the accountability is likely to be much higher.

Besides other factors, technical excellence is also the product of a psychologically safe work environment. Organizations should empower their software development teams to take initiatives, experiment, learn, sometimes even fail, then adapt. Technical excellence is, to some extent, based on the knowledge that comes from lived experiences. What works for team A may not necessarily work for team B. Empiricism occurs by experimentation, adaptation and consequently growth. Simple behaviors such as avoiding blame, building trust, and discouraging negativity can promote psychological safety.

Support from the leadership and the customer are essential and valuable enablers for technical excellence. Not only does it foster positive relationships between the development team and the leadership and the customer, but it also ultimately empowers team members to develop their skills and enables them to work autonomously to meet the goals set for technical excellence. For example, our participants explained that when the customer is "on the same page", the team feels "safe" to invest in technical excellence.

Knowledge and growth do not advance without ongoing training. Software engineering is an evolving practice. Therefore, investing in upskilling the development team will bring new knowledge to the team and equip its members with up-to-date engineering practices.

Team

Despite being rooted in some individual qualities, i.e., motivation, curiosity, and self-development, at the end of the day, technical excellence is a collective endeavor. The collective behavior of the team insinuates intense collaboration that will facilitate knowledge sharing. The pursuit of technical excellence is not a "one-man army" either. Software development is a highly collaborative activity, and the output is a team effort. Team members should support each other which implies collective efforts for solving complex obstacles.

The customer does not always understand software code, architecture and design concepts like scalability. They are motivated by the business value of the product. For a better understanding of each other's values and goals, software teams should establish a dialogue with their customers. This dialogue should be educational, aiming at creating an agreement on what constitutes a business value.

Individual

Individuals fuel the pursuit of technical excellence. This fuel is sourced from the individual's motivation, intellectual curiosity, and the desire for self-development. Although our data show that developing these qualities rest on the individual, e.g., "starts from the individual" (P12), it also indicates that other influencers such as peer pressure, e.g., "absorb the energy somehow from the others" (P12) lead to a change of behavior. Team members are motivated to adhere to the group values. As adhering to the group norms and values enhances the social connection of the individual with the group and increases the likelihood of acceptance and the feeling of belonging [65].

6.2. Implication on research

As shown in the related work discussion (Section 2, technical excellence in software engineering is a forgotten topic. Engineering excellence is an integral part of the craft of engineering. The quest of being excellent at software engineering is not only rewarding for the outcome, i.e., the product or the software, it is also a pursuit for the team technical growth, i.e, betterment of their practices and knowledge. This area of research should receive more merit and attention. Our study opens up at least five areas for future research to complement and extend our work.

The prevalence of our findings. Qualitative studies seek to reveal in-depth and rich conclusions with a small sample [66]. Our conclusions are based on agile practitioners' experiences and constrained by the inherent limitations of qualitative research, e.g., a small sample and not aiming for statistical generalization. We still do not know the prevalence of these findings industry-wide. Future work could investigate the prevalence of our participants' interpretations of *Principle 9*, the adoption of practices found in response to **RQ2**, and the salience of enabling conditions presented under **RQ3** in the industry.

Successful and failed cases. Case studies increase our knowledge of a particular phenomenon in depth [67,68]. While a quantitative study may broaden our knowledge of the prevalence of the interpretation, the practices and the enablers we discussed in this paper, it will be short of providing insights into the struggles and difficulties of software teams and their organizations in their pursuit of building an environment where technical excellence is feasible. Future work could investigate successful and failed cases in order to draw lessons on what

works and what does not. By studying positive and negative cases, our understanding of a particular phenomenon would be further enhanced, and the validity of the conclusions would be strengthened [69].

Mindset and formal architecture processes. While software architecture research focuses on formal processes led by dedicated architects [15,48,49], our study shows a complementary way of developing architecturally sound software, namely by promoting a mindset and the skills that lead developers to focus on sustainable code. Future research could examine how and when these two strategies can be fruitfully combined. A valuable point of departure to this end could be existing work on agile architecture and the forces affecting the choice of different architectural processes [58]. Moreover, future research could develop and empirically examine different strategies for promoting a mindset of continuous attention to sustainable code.

Mindset in process improvement and learning. Despite the abundant research on software process improvement [16,32,51,52] and on learning [42,70] in software development, these research streams rarely account for the role of mindsets as enablers of learning processes. Our study shows that a growth mindset, a continuous improvement mindset, and a team mindset can play important roles for achieving technical excellence. Future research could examine strategies for promoting these mindsets and the effects of these mindsets on process improvement and learning in software development teams.

Strategies for leeway. A key theme of our study was the importance of giving developers leeway for caring about architecture and learning. Although our study points to some strategies to provide leeway, future research could design or explore such strategies more comprehensively. For example, some companies allow employees to use some part of their work time for own initiatives. Future research could explore how this and another strategies for leeway affect architecting and learning in software development.

Intriguingly, other potential research questions present themselves. For example, how knowledge accumulated in the pursuit of technical excellence is maintained in software teams? How project constraints (e.g., budget, schedule, etc.) influence the willingly agreed excellence standards in software teams? How a technical excellence community of practice can be implemented to promote and sustain excellence in software development for organizations? Etc. Our research community has many opportunities to influence this area of research.

7. Conclusion

The journey towards technical excellence in agile software development started with the proposition of *Principle 9* in the agile manifesto. Although abstract, the principle was sufficient for practitioners to embrace it, figure out an interpretation, then leverage other agile values to ensure continuity and the development of its implementation. Our findings show that practitioners turned the proposition of *Principle 9* into a motivation and an interest to deliver technically superior software. The commitment, which grows when the social setting facilitates conditions to nurture it, is at the heart of this journey.

CRediT authorship contribution statement

Adam Alami: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Oliver Krancher:** Formal analysis, Writing – review & editing. **Maria Paasivaara:** Formal analysis, Writing – review & editing.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.infsof.2022.106959.

Acknowledgments

We would like to thank our interviewees and the focus group participants for their time and effort in making this study possible. We would also like to thank two anonymous reviewers for their excellent and highly constructive comments.

References

[1] C.G. Cobb, The Project Manager's Guide to Mastering Agile: Principles and Practices for an Adaptive Approach, John Wiley & Sons, 2015.
[2] P. Hohl, J. Klünder, A. van Bennekum, R. Lockard, J. Gifford, J. Münch, M. Stupperich, K. Schneider, Back to the future: origins and directions of the "agile manifesto"–views of the originators, J. Softw. Eng. Res. Dev. 6 (1) (2018) 1–27.
[3] A. Cockburn, Agile software development joins the" would-be" crowd, Cutter IT J. 15 (1) (2002) 6–12.
[4] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley Professional, 2000.
[5] B. Boehm, Get ready for agile methods, with care, Computer 35 (1) (2002) 64–69.
[6] M. Fowler, J. Highsmith, et al., The agile manifesto, Softw. Dev. 9 (8) (2001) 28–35.
[7] M. Laanti, J. Similä, P. Abrahamsson, Definitions of agile software development and agility, in: European Conference on Software Process Improvement, Springer, 2013, pp. 247–258.
[8] T. Dingsøyr, S. Nerur, V. Balijepally, N.B. Moe, A Decade of Agile Methodologies: Towards Explaining Agile Software Development, Elsevier, 2012.
[9] B. Latte, S. Henning, M. Wojcieszak, Clean code: On the use of practices and tools to produce maintainable code for long-living, 2019.
[10] R.C. Martin, Clean Code-Refactoring, Patterns, Testen und Techniken für sauberen Code: Deutsche Ausgabe, MITP-Verlags GmbH & Co. KG, 2013.
[11] R.C. Martin, Clean Architecture: a Craftsman's Guide to Software Structure and Design, Prentice Hall, 2018.
[12] P. McBreen, Software Craftsmanship: the New Imperative, Addison-Wesley Professional, 2002.
[13] B. Pyritz, Craftsmanship versus engineering: Computer programming—An art or a science? Bell Labs Tech. J. 8 (3) (2003) 101–104.
[14] B. Boehm, R. Turner, Using risk to balance agile and plan-driven methods, Computer 36 (6) (2003) 57–66.
[15] P. Kruchten, H. Obbink, J. Stafford, The past, present, and future for software architecture, IEEE Softw. 23 (2) (2006) 22–30.
[16] A.A. Khan, J. Keung, M. Niazi, S. Hussain, A. Ahmad, Systematic literature review and empirical investigation of barriers to process improvement in global software development: Client–vendor perspective, Inf. Softw. Technol. 87 (2017) 180–205.
[17] M.J. Parzinger, R. Nath, A study of the relationships between total quality management implementation factors and software quality, Total Qual. Manag. 11 (3) (2000) 353–371.
[18] A. Alami, M. Paasivaara, How do agile practitioners interpret and foster "technical excellence"? in: Evaluation and Assessment in Software Engineering, 2021, pp. 10–19.
[19] J.E. Hannay, T. Dybå, E. Arisholm, D.I. Sjøberg, The effectiveness of pair programming: A meta-analysis, Inf. Softw. Technol. 51 (7) (2009) 1110–1122.
[20] O.P. Timperi, An overview of quality assurance practices in agile methodologies, in: Seminar in Software Engineering, 2004.
[21] L. Madeyski, Test-Driven Development: An Empirical Evaluation of Agile Practice, Springer Science & Business Media, 2009.
[22] G. Arcos-Medina, D. Mauricio, Aspects of software quality applied to the process of agile software development: a systematic literature review, Int. J. Syst. Assur. Eng. Manag. 10 (5) (2019) 867–897.
[23] L. Prechelt, H. Schmeisky, F. Zieris, Quality experience: a grounded theory of successful agile projects without dedicated testers, in: 2016 IEEE/ACM 38th International Conference on Software Engineering, ICSE, IEEE, 2016, pp. 1017–1027.
[24] J.F. Tripp, D.J. Armstrong, Exploring the relationship between organizational adoption motives and the tailoring of agile methods, in: 2014 47th Hawaii International Conference on System Sciences, IEEE, 2014, pp. 4799–4806.
[25] T. Dybå, T. Dingsøyr, Empirical studies of agile software development: A systematic review, Inf. Softw. Technol. 50 (9–10) (2008) 833–859.
[26] R. Hoda, N. Salleh, J. Grundy, The rise and evolution of agile software development, IEEE Softw. 35 (5) (2018) 58–63.
[27] C. Yang, P. Liang, P. Avgeriou, A systematic mapping study on the combination of software architecture and agile development, J. Syst. Softw. 111 (2016) 157–184.
[28] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley Professional, 2018.
[29] A. Martini, J. Bosch, A multiple case study of continuous architecting in large agile companies: current gaps and the CAFFEA framework, in: 2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA, IEEE, 2016, pp. 1–10.

[30] J. Madison, Agile architecture interactions, IEEE Softw. 27 (2) (2010) 41–48.
[31] N. Ramasubbu, S. Mithas, M.S. Krishnan, C.F. Kemerer, Work dispersion, process-based learning, and offshore software development performance, MIS Q. (2008) 437–458.
[32] T. Dingsøyr, M. Mikalsen, A. Solem, K. Vestues, Learning in the large-an exploratory study of retrospectives in large-scale agile development, in: International Conference on Agile Software Development, Springer, Cham, 2018, pp. 191–198.
[33] O. Salo, P. Abrahamsson, An iterative improvement process for agile software development, Softw. Process Improv. Pract. 12 (1) (2007) 81–100.
[34] Q. Patton, Qualitative Research and Evaluation Methods, third ed. M, Sage, Thousand Oaks, 2005.
[35] J.R. Wolgemuth, Z. Erdil-Moody, T. Opsal, J.E. Cross, T. Kaanta, E.M. Dickmann, S. Colomer, Participants' experiences of the qualitative interview: Considering the importance of research paradigms, Qual. Res. 15 (3) (2015) 351–372.
[36] D.S. Cruzes, T. Dyba, Recommended steps for thematic synthesis in software engineering, in: 2011 International Symposium on Empirical Software Engineering and Measurement, IEEE, 2011, pp. 275–284.
[37] B. Saunders, J. Sim, T. Kingstone, S. Baker, J. Waterfield, B. Bartlam, H. Burroughs, C. Jinks, Saturation in qualitative research: exploring its conceptualization and operationalization, Qual. Quant. 52 (4) (2018) 1893–1907.
[38] C.C. Venters, C. Jay, L. Lau, M.K. Griffiths, V. Holmes, R.R. Ward, J. Austin, C.E. Dibsdale, J. Xu, Software sustainability: The modern tower of babel, in: CEUR Workshop Proceedings, Vol. 1216, CEUR, 2014, pp. 7–12.
[39] C. Dweck, What having a "growth mindset" actually means, Harv. Bus. Rev. 13 (2016) 213–226.
[40] A. Silva, T. Araújo, J. Nunes, M. Perkusich, E. Dilorenzo, H. Almeida, A. Perkusich, A systematic review on the use of definition of done on agile software development projects, in: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, 2017, pp. 364–373.
[41] T. Mårtensson, D. Ståhl, A. Martini, J. Bosch, Continuous architecture: Towards the goldilocks zone and away from vicious circles, in: 2019 IEEE International Conference on Software Architecture, ICSA, IEEE, 2019, pp. 131–140.
[42] O. Krancher, P. Luther, M. Jost, Key affordances of platform-as-a-service: self-organization and continuous feedback, J. Manage. Inf. Syst. 35 (3) (2018) 776–812.
[43] B. Fitzgerald, K.-J. Stol, Continuous software engineering: A roadmap and agenda, J. Syst. Softw. 123 (2017) 176–189.
[44] A. Edmondson, Psychological safety and learning behavior in work teams, Adm. Sci. Q. 44 (2) (1999) 350–383.
[45] P. Kruchten, What do software architects really do? J. Syst. Softw. 81 (12) (2008) 2413–2416.
[46] R. Faber, Architects as service providers, IEEE Softw. 27 (2) (2010) 33–40.
[47] I. Iso, IEC 9126–software engineering–product quality, Int. Organ. Stand. 43 (2001) 59–60.
[48] S. Blair, R. Watt, T. Cull, Responsibility-driven architecture, IEEE Softw. 27 (2) (2010) 26–32.
[49] J. Díaz, J. Pérez, J. Garbajosa, Agile product-line architecting in practice: A case study in smart grids, Inf. Softw. Technol. 56 (7) (2014) 727–748.
[50] P. Kruchten, The software architect, in: Working Conference on Software Architecture, Springer, 1999, pp. 565–583.
[51] M. Sulayman, C. Urquhart, E. Mendes, S. Seidel, Software process improvement success factors for small and medium web companies: A qualitative study, Inf. Softw. Technol. 54 (5) (2012) 479–500.
[52] M.A. Ringstad, T. Dingsøyr, N. Brede Moe, Agile process improvement: diagnosis and planning to improve teamwork, in: European Conference on Software Process Improvement, Springer, 2011, pp. 167–178.
[53] Y. Andriyani, R. Hoda, R. Amor, Reflection in agile retrospectives, in: International Conference on Agile Software Development, Springer, Cham, 2017, pp. 3–19.
[54] I. Allison, Y. Merali, Software process improvement as emergent change: A structurational analysis, Inf. Softw. Technol. 49 (6) (2007) 668–681.
[55] S. McIntosh, Y. Kamei, B. Adams, A.E. Hassan, The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects, in: Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 192–201.
[56] J. Humble, J. Molesky, Why enterprises must adopt devops to enable continuous delivery, Cutter IT J. 24 (8) (2011) 6.
[57] R.L. Nord, J.E. Tomayko, Software architecture-centric methods and agile development, IEEE Softw. 23 (2) (2006) 47–53.
[58] M. Waterman, J. Noble, G. Allan, How much up-front? A grounded theory of agile architecture, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1, IEEE, 2015, pp. 347–357.
[59] O. Krancher, Agile software development practices and success in outsourced projects: The moderating role of requirements risk, in: International Conference on Agile Software Development, Springer, 2020, pp. 56–72.
[60] Y. Dittrich, S. Vaucouleur, S. Giff, ERP customization as software engineering: Knowledge sharing and cooperation, IEEE Softw. 26 (6) (2009) 41–47.
[61] R. Sharma, P. Yetton, The contingent effects of management support and task interdependence on successful information systems implementation, MIS Q. (2003) 533–556.
[62] N.P. Repenning, J.D. Sterman, Capability traps and self-confirming attribution errors in the dynamics of process improvement, Adm. Sci. Q. 47 (2) (2002) 265–295.
[63] H. Goldblatt, O. Karnieli-Miller, M. Neumann, Sharing qualitative research findings with participants: Study experiences of methodological and ethical dilemmas, Patient Educ. Couns. 82 (3) (2011) 389–395.
[64] T.A. Schwandt, The Sage Dictionary of Qualitative Inquiry, Sage publications, 2014.
[65] S. Gavrilets, P.J. Richerson, Collective action and the evolution of social norm internalization, Proc. Natl. Acad. Sci. 114 (23) (2017) 6068–6073.
[66] M.B. Miles, A.M. Huberman, J. Saldana, et al., Qualitative Data Analysis: A Methods Sourcebook, Sage, Thousand Oaks, CA, 2014.
[67] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2) (2009) 131–164.
[68] R.K. Yin, The case study anthology, Sage, 2004.
[69] A. Hanson, Negative case analysis, Int. Encycl. Commun. Res. Methods (2017) 1–2.
[70] O. Krancher, S. Slaughter, Governing individual learning in the transition phase of software maintenance offshoring: A dynamic perspective, in: 2013 46th Hawaii International Conference on System Sciences, IEEE, 2013, pp. 3543–3552.