How Do Agile Practitioners Interpret and Foster "Technical Excellence"?

Adam Alami
IT University of Copenhagen
Denmark

Maria Paasivaara
LUT University, IT University of Copenhagen & Aalto University
Finland

ABSTRACT

"Technical excellence" is a nebulous term in agile software development. This vagueness is risky, as it creates a gap in the understanding of agile that may have consequences on how software development practitioners operate. Technical excellence is the only reference to quality in the agile manifesto. Hence, it is fundamental to understand how agile software development practitioners both interpret and implement it. We conducted interviews with twenty agile practitioners about their understanding of the term "technical excellence" and how they approach the task of fostering it. To validate the findings, two focus group meetings were conducted after the interviews and the analysis of the data were completed. We found that the configuration of technical excellence is made of four traits: (1) software craftsmanship; (2) software quality (3) mindset for excellence; and (4) consistency with good software engineering practices. Fostering technical excellence is a continuous endeavor. Further, we identified three key principles that were commonly cited as essential to implementing technical excellence, namely: 1) continuous learning; 2) continuous improvement; and 3) control of excellence. Based on our findings, we present several recommendations for software development teams seeking to better realize the goal of technical excellence in their agile implementation.

KEYWORDS

Agile, Software Development Methods, Technical Excellence, Agile Principles

1 INTRODUCTION

Agile is a software development philosophy with implications for effective and maneuverable software development. Boehm [7] describes agile methods as the result of rapid prototyping and development, viewing software programming as a craft rather than an industrial process. The agile methods include iterative, incremental, self-organizing, and emergent concepts. In this report, we use the term "agile" to refer to software development methodologies as defined by Cockburn [11] being light, remaining maneuverable but with sufficient rules of project behavior.

Agile was founded on twelve principles, outlined in the Agile Manifesto [17]. Identifying the roots and the meanings of these principles is important in understanding agile software development. The highest priority is to satisfy the customer through early and continuous delivery of valuable software, to welcome changing requirements, and to deliver working software frequently. For example, Principle 1 is to deliver something to the end user, and Principle 2 is to measure the added value to the user. Daily builds are important, and simplicity and quality of design are critical.

One common criticism of the Agile Manifesto is that it is too vague [23]. Since it is an abstract concept, different teams can mean different things when they use the term "agile"; therefore, the best option is for teams to affirm that they have a shared understanding of the agile terms early in the development process [23]. This practice is particularly important with the ninth principle, stated below, which emphasizes focus on technical excellence and good design. This principle, the only reference to quality in the manifesto, is a recognition that agile software development teams cannot rely solely on abstract principles to achieve high-quality products.

Principle 9:
"Continuous attention to technical excellence and good design enhances agility." [17]

Hohl et al. [21] argue that the idea behind agile is still valid and relevant. However, they point out that the misunderstanding of the agile manifesto is a common occurrence. They explain that it leads to "unpleasant effect" of agile being misinterpreted, which leads to pressure on the developers instead of promoting a motivating and productive work space. These misunderstandings of the concept also lead to a poor implementation of agile [21].

Hohl et al. [21] also investigated the future directions of the manifesto. They conducted several interviews and workshops with the original founders of the manifesto and other practitioners. They are concerned that the manifesto's values and principles end up being ignored. People use the principles without understanding them [21].

Dingsøyr et al. [14] examined agile software development literature to illustrate how the research on agile has progressed during the 10 years following the articulation of the manifesto. They concluded that more work has to be done to investigate the principles of agile that are at once unequivocal and useful for practice [14]. They further advocate empirical studies on the "core" of agile values and principles [14].

A principle is a statement of fundamental truths or general laws. It is not directly associated with a technology, method, or technique [41]. As such, Principle 9 triggers more questions than it provides answers. Do agile teams have a common understanding of "technical excellence"? And, do they have common strategies for implementing it? We propose the following research questions:

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

RQ2: *How do agile practitioners foster "technical excellence"?*

Adam Alami and Maria Paasivaara. 2021. How Do Agile Practitioners Interpret and Foster "Technical Excellence"?. In Evaluation and Assessment in Software Engineering (EASE 2021), June 21–23, 2021, Trondheim, Norway. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3463274.3463322.

We sought to investigate how agile software development practitioners understand and interpret Principle 9 of the agile manifesto. We propose guidelines to operationalize the term "technical excellence" based on the practices and values of people in the field. Agile practitioners should make these implicit shared understandings explicit in their development team to better realize the potential of agile. Our work, to the best of our knowledge, is the first to investigate "technical excellence" in agile empirically; while previous work is either practitioner literature (e.g., [27–29]) or not empirically founded (e.g., [36]).

2 RELATED WORK

Our search of the agile software development literature shows a mounting interest in various topics related to quality in agile. However, the current work is not directly linked to Principle 9 of the manifesto. It investigates various topics in isolation; like pair programming (e.g., [10, 19, 22]), refactoring (e.g., [16, 30, 32]) and test-driven development (e.g., [9, 26, 33]). Whether these practices are influenced by Principle 9 is still not well understood.

Recent studies call for more empirical investigation of the topic of technical excellence [4, 15, 37, 43]. This is partly because most existing work is practitioner literature (e.g., [27, 28]) and there is a need for further empirical support of the current work. "There are no empirical studies concerning the lack of acceptance of clean Code in practice" [37]. "There does not appear to be much research exploring the techniques that software developers actually use in their day-to-day practice and which of these techniques they find most useful" [43].

There is no existing work on the interpretation of "technical excellence." However, there is work on the following sub-topics: (1) software craftsmanship, (2) quality in agile and (3) the need to define "technical excellence" in agile. We present and discuss the related work of these topics in this section.

2.1 Software Craftsmanship

An important fact is that people write software. As discussed by Pyritz [36], 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 [36].

2.2 Quality in Agile

Arcos-Medina and Mauricio conducted a systematic review of agile literature on quality. They identified a catalog of factors, agile practices, and metrics that influence quality in agile methods [3]. Five critical success factors were proposed: teamwork practices, engineering practices, management practices, documentation practices and testing practices. Teamwork practices represent 24% of the available literature, while management practices account also for 24%. However, in most studies identified in this review, quality was not the primary topic of investigation. Topics such as "success factors" and "process optimization" were examined. This literature study shows that we still need empirical work to investigate agile traits and their correlation with achieving quality.

Timperi [45] discusses several engineering practices used by agile team to assure quality, such as, inspections, pair programming, test-driven development, coding standards, collective code ownership, and refactoring [45].

Dingsøyr et al. [14] 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. Structuring naming, and documentation were mentioned often as well. Code comprehension was emphasized numerous times, as well as debugging. Code smells were mentioned, such as duplication, wrong abstraction, wrong naming, missing tests, side effects, high number of parameters, noise, and cycles. Documentation was mentioned, as was principles such as DRY (don't repeat yourself) and KISS (keep it simple stupid) [14].

Prechelt et al. propose "quality experience" [35], 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 rapidly repairs deficiencies when they occur" [35]. This work shows that quality in agile teams is assured by a combination of technical (e.g., modular architecture) and non-technical (e.g., high motivation, held responsible and feel responsible) factors [35].

2.3 The need to define "technical excellence" in agile

Tripp and Armstrong [46] 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 [46]. If the growing interest in agile is partly driven by achieving quality, then how does the agile manifesto advocate for quality and excellence? Our study is an empirical investigation on how agile practitioners interpret and foster "technical excellence."

Recent reviews of the literature [15, 20] show limited interest on the topic of quality in agile. Since agile influences the value stream of software teams, there is a merit to evaluate how agile teams achieve quality and technical excellence.

3 METHODS

We opted to answer our research questions using agile practitioners' experiences. Hence, we chose a qualitative method, which allows us to explore practitioners' experience to gain a depth of understanding about technical excellence (TE) in agile methods in order to

propose a set of 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 [48]. Practitioners' experience is relevant to investigate our research questions. We explicitly looked for participants who took actions to implement TE in their respective teams, i.e., to understand the world as others experience it.

To permit a degree of flexibility in our conversations and questioning, 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.

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 enhances 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 1: Key parts of the interview questions

Subject Selection. We interviewed 20 agile practitioners who we recruited from LinkedIn, and whose experience in software development environment 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 background in software development and having interest and experience in implementing technical excellence. As eligible participants were identified those who had a minimum of five years of experience in agile software development, and who had started their careers as software developers and actively participated in implementing and fostering TE 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 development industry and "Dev. Exp." shows the software development experience of the participants.

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 minutes and

#	Role	Agile Set-up	Exp. in Sw. Industry	Dev. Exp.
P1	Sr. Agile Product Manager	Scrum	20	12
P2	Agile Coach	Scrum	20	14
P3	Agile Coach	Scrum	20	12
P4	Agile Delivery Specialist	Scrum	12	8
P5	Scrum Master	Scrum	14	10
P6	Scrum Master / Team Agilist	Scrum	15	10
P7	Agile Coach	Scrum	18	12
P8	Project Manager	Scrum	6	4
P9	Project Manager	Scrum	28	12
P10	Portfolio Manager	Scrum	20	14
P11	Program Manager	Scrum	21	13
P12	Scrum Master	Scrum	31	16
P13	Senior Product Manager	Scrum	18	14
P14	Project Manager	Scrum	15	10
P15	Head Of Quality Assurance	Scrum	14	7
P16	Product Owner	Scrum	11	7
P17	Lead QA Engineer	Scrum	10	6
P18	Project Manager	Scrum	8	5
P19	Agile Coach	Scrum	7	5
P20	Scrum Master	Scrum	7	5

Table 2: The Study Interviewees

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 results of this method are not always accurate. Hence, some manual corrections were necessary. Once all interviews were transcribed, we sent the transcripts to the participants for review (see section 5).

Data Analysis. We used thematic coding to analyze the data, following the guidelines by Cruzes and Dybå[13]. Our approach is inductive, codes are derived from the data. 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 in categories by similarity. Table 3 shows examples of our codes and their corresponding themes. We initiated the coding process and analysis as soon as interview transcripts become available so that we could monitor for saturation. We reached saturation [38] after twelve interviews, as no new additional codes, themes, or information emerged from the data after that. However, we opted to interview the remaining eight participants. The coding was performed by the first author. We used peer debriefing sessions to critically review the codes and their categorization. During these sessions, the second author provided feedback on the coding decisions. This has been done iteratively until a consensus was reached on a final list of codes and themes.

Member Validation. We opted to use focus groups for member checking, a technique for the validation of the results. During these focus groups, we presented our findings to the participants to check for accuracy and resonance with their experiences. We invited all

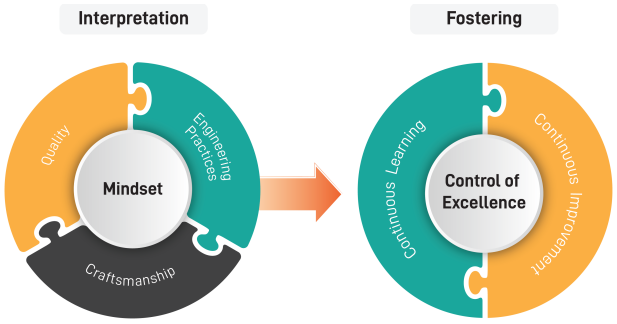

Figure 1: The Interpretation and Fostering of Technical Excellence

participants to this validation exercise, however, only six participants accepted. Due to their availability, we had to schedule two separate focus group meeting instances (see section 5).

4 FINDINGS

Our data show that there is a strong awareness of technical excellence amongst our participants, as well as a moderate degree of consensus in the definition of this term. Technical excellence is cited by our participants as a core agile value, and some participants further claimed that there is no agile without technical excellence. Our participants describe technical excellence as consisting of both engineering and non-engineering components. According to our participants, agile is an empirical process wherein technical excellence occurs through experimenting, learning, and improving.

Figure 1 illustrates our findings. The mindset for excellence is at the centre of interpretation of technical excellence. Our participants pointed out that technical excellence starts with the right mindset and it is a continuous endeavor. It is the ongoing improvement of practices and processes through learning, re-examining and the incremental implementation of measures to get better at developing software.

In response to **RQ1**, our participants interpret "technical excellence" being the joining of four traits that are individually distinct: (1) software craftsmanship, (2) software quality, (3) mindset and (4) engineering practices. Table 3 shows the codes and their occurrences in data, the number of times the code was talked about and discussed by a participant. In the upcoming subsections, we emphasize the definitions using light gray for the text background and boxes for recommendations.

4.1 Software Craftsmanship

Our participants define software craftsmanship as the ability to create elegant and high-quality code that delivers significant value and can be scaled up sustainably. P4 explains: *"Technical excellence is also achieved by craftsmanship and creativity."* P3 describes how 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. [...] So that's also one of the craftsmanship element."* He further explains: *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. My work is acknowledged, my work adds value to the real world. I'm doing something for the betterment [...] when you have that mindset in your developer, he's going to take care of quality himself as well ... So, it all comes under this umbrella of continuous attention to detail. And how do you enable that is, again, by the craftsmanship element when the developer is practicing mastery be it a QA, be it a developer, be it an architect, is proud of their work."*

> **Software Craftsmanship:** Our participants perceive software development to be like a craft and the process of developing software to be the product of a craftsman. This implies that developers in agile have skills and perform their jobs with precision. They are motivated by mastery rather than simply writing code.

Then, what is the implication of this finding? Junior team members should spend a significant time in an apprenticeship learning period to acquire the craft and the mastery for writing high-quality code, under the tutelage of a senior member who has shown great skill or proficiency in writing code. P12 shared her company's experience with us:

> *"Another example, is when a new team member joins, we check the quality of their code. If it is not within our expectations, then he is assigned to a mentor for a period of time until he learns the craft of writing good software code. I know, it is a loss of a resource for some time but it is worth the investment and the long term reward. Because, what is the point of having someone who write a code below our standards! We mitigate the risk of having under-performing members by investing in teaching them the skills."*

Writing code has intangible qualities which should be passed on from one generation to another to safeguard this heritage within the organization.

> **Recommendation 1:** Craftsmanship should be internalised through apprenticeship. Organizations should offer apprenticeship aimed at facilitating internalization through the establishment of peer-to-peer learning.

Discussion of craftsmanship. Few authors have highlighted the quality of craftsmanship in software development; those who have [29, 36] explain this by pointing to the early influence of manufacturing procedures and techniques on the development of software engineering. However, according to our participants, software development is a creative task. This creativity should be nurtured and passed on from one generation to another.

Pyritz [36] claims that the idea of just following the directions in the assembly line fails with software. Software development is a mental activity that requires creativity and problem-solving skills. People are the most critical resource in the process, and human beings can accomplish the same things in various ways [36]. In a similar vein, McBreen [29] suggests that having good, experienced developers is the key to creating quality software in a short time. Good developers are those who continually learn and get better at their craft. They have enthusiasm for the work, and they pass this enthusiasm to other members of the team. Good developers use many technologies and platforms, but they become

Theme	Code	N	Example of verbatim
The configuration of TE	**Software craftsmanship**	**5**	*"Creating clean code and sustainable design is a craft."* P4.
	Quality	**11**	*"Flexible, scalable, and open to change. This is technical excellence, delivering high-quality software by ensuring flexible, scalable and open to change design, with clean code."* P20.
	Mindset	**5**	*"That's the first thing you need to look for. A mindset attitude. It's soft. So, they are personal traits, openness, flexibility."* P10.
	Engineering practices	**9**	*"I truly believe that this particular statement is the very foundation or a founding pillar of the whole idea of agility. And this actually highlights the engineering aspect of agile."* P3.
The development of TE	**Continuous learning**	**12**	*"For me continuous attention to technical excellence is actually just an experiment and learn all the time. So, experiment, make mistakes and learn from the process. This is continuous attention. And share your mistakes and frustration with the team."* P13.
	Continuous improvement	**15**	*"Agile is also about continuous learning and improvement. Each iteration is a learning opportunity. We learn and then we take actions to improve. I like to emphasize this. You need openness and empowering in your team in order to achieve this continuous improvement."* P4.
	Control of excellence	**6**	*"We control technical excellence via the Definition of Done, but as I mentioned before, we need to invest on the people. We need to coach them to be technically excellent."* P4.

Table 3: Example themes and codes (N shows the number of mentions in total in the data)

productive quickly. McBreen also indicates that talent should not be confused with mastery; mastery is the ability to create and enhance robust, high quality applications while nurturing other developers, and may take 15 years to develop. To foster craftsmanship, master developers should choose their own apprentices and journeymen, as the cultivation of other developers is a long-term commitment [29].

Software craftsmanship is an alternative craft model that places people at the center of the process. The craftmanship model of software development follows the phases of apprenticeship, journeymanship, and practice until the individual becomes a master craftsman. The training occurs over several years on the job, under the tutelage of a master craftsman, wherein the apprentice is held to the highest standards before moving to the next phase to become a craftsman [36]. The software master craftsman should have the following qualifications: 1) a proven track record of successful applications; 2) sound work habits, such as accepting responsibility and completing tasks; and 3) detailed working knowledge of the latest technologies, tools, processes, and practices (including awareness of those that should be avoided). In addition, a master craftsman should work well with other people, have expert domain knowledge, take a personal approach to problem-solving, and love writing software [29].

Apprentices should be chosen by the craftsman to join their team. The apprentice will work with the journeyman and craftsman for a period typically lasting five years. The apprentice can become a journeyman upon consistent mastery of his or her assigned work. The journeyman becomes responsible for more complex tasks and part of the training of apprentices through a pair-programming approach. The journeyman may journey to work with other craftsman for brief periods of time. The end of a phase should be celebrated [29].

4.2 Software Quality

"Quality" was extensively referred to by our participants. No single definition achieved saturation, but two attributes emerged from interviews: (1) clean code, and (2) sustainable software design.

Clean code. The term "clean code" refers to code that can be easily understood and easily changed. P9 explained that *"the key to writing good clean code is to know how to avoid complexity."* P19 asserts that complexity of code hinders scalability. He stated: *"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."*

Clean code can be understood whether the reader is the author of the code or someone totally unfamiliar with the code. The code's meaning is so clear and concise that it does not require anything new to clarify it, and it is difficult to misunderstand. Our participants call for simplicity in writing software code, with remarks such as: *"A good design is usually a simple design"* (P1); *"Simple code allows maintaining, debugging, refactoring, and adding features with limited knowledge of how the entire system works"* (P20).

Clean code can not only be easily understood, but it can be easily changed as well. It can be extended and/or refactored, and it is easy to fix bugs in the codebase. The code is so clear that the reader can understand the code well enough to make changes without fear of breaking any existing functionality. Those who read the code feel confident enough to make necessary changes. *"Agile is about delivering efficient code, easy to change for future requirements. This is agility"* (P4). Even if developers write code for machines to execute tasks, the code should be readable to humans first. P20 stated: *"Clean code is about recognizing that your audience isn't just a computer, it's real humans!"*

Technical excellence advocates for clean code. Writing clean code might cost time, but it is an investment in knowledge creation, agility and the evolution of the software. It creates easily maintainable, extensible code, easy to debug and refactor.

Discussion of clean code. If code is hard to understand then a bad fix is likely, resulting in increasing number of bugs. Cultural moves in agile software toward DevOps and combining it with clean code and enforcement of quality via build pipelines and review processes will lead to better systems. Selecting or authoring code style guides that are reviewed and agreed on by the core team is helpful. These guides include code formatting, consistent use of language features, naming conventions, and use of meaningful names. Review by a second developer is also helpful. If problems are identified, discussions between the two can resolve the problem. This requires openness and trust between the parties, which can be a long learning process [25].

Latte et al. [25] found measures to lower complexity in learning and communication, and these require a culture of openness. The choice of procedures and practices is more a cultural than a technical issue. LaToza and Venolia [24] explain that developers routinely face problems with understanding code written by others, including the rationale behind the coding, the history of a piece of code, and any code not written recently. Developers go to great lengths to create and maintain rich mental models of code, understand the rationale behind the code, and determining tools and work practices. Understanding the rationale behind code is the most serious problem developers face, but the code itself is – or should be – the best source of information about the code. Using design documents to understand code is not often practiced, both because locating the design documents can be difficult and because the documents are often not worth looking for. In addition, design documents are not often updated. Therefore, asking a teammate is the next step in understanding the rationale, but each unplanned meeting represents an interruption and recovering from interruptions can be difficult [24].

Sustainable software design. P1 claims that agile way builds high-quality software that has sustainable design. He states, *"if you would implement an agile working style, the goal would be to build software that has high-quality, so it can be changed in the future in a sustainable way."* Software that has sustainable design allows developers to deliver modifications to the customer quickly with fewer bugs and at a lower cost of ownership. Increased business agility also results from sustainable software development. P20 explains this concept: *"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."* The quality of the software code must evolve gracefully with every change. P3 explains: *"The quality of the code should not decline with every change."* Our participants consider agile software development as a process which aims to create reliable, long lasting software that meets the needs of users while maintaining quality. Sustainable software is delivering customer value today and tomorrow.

Software Quality: Our participants equate software quality with the software code being clean and the software design being sustainable.

Software quality is a complex concept to define in the context of software development [2, 31]. Software developers aim to achieve quality software, but the complexity of the effort makes it difficult to determine when the software is, in fact, a quality product [34]. Our participants seem to have a rather narrow view of quality. They equate the concept with clean code and sustainable software design. This treatment of quality is not wholesome. This could be because principle 9 has a clear scope, which is the software code and design. It is clear that code quality influences product quality. Software development teams should treat the code as an inseparable part of the product.

Then, what is the implication of this finding? To enable business agility, organizations should invest in achieving software quality. Quality (clean code and sustainable software) makes agility possible by producing a reusable code and a sustainable design. P6 explains:

> *"Quality is code and design. I'd like to illustrate that with the Lego analogy. I'm thinking in Lego bricks, you can only add a new Lego brick because in your system it will be set that I need to be OK if you set a new brick. And nothing breaks ... That's the idea. Work like in the Lego brick model design or something that works for you, for your organization future. That's how you enhance the agility of your customers' business. You enable them to continuously come up with changes and your Lego model can handle it and doesn't break."*

According to this testimony, software quality adds value to the business.

Recommendation 2: Developers must learn how to write clean code and design software that can support future changes. These qualities could be promoted as values in the agile team, which should influence team behavior.

Discussion of sustainable software design. Venters et al. [47] define sustainability as the quality of being supported, capable of being endured, or continuing to exist. The notion of being "supported" also lends itself to the theory that sustainability includes the concepts of longevity and maintenance. The most widely adopted definition, proposed by the Brundtland commission, has defined sustainability as meeting the needs of the present without compromising the future generations' ability to meet their needs [47].

Seacord et al. [40] define software sustainability as the ability to modify a system based on customer needs. Although they use sustainment and maintenance interchangeably, the IEEE definitions differentiate between these two terms [12]. Maintenance refers to the process of modifying a system after delivery, while sustainment refers to meeting a range of stakeholders' needs.

4.3 Mindset

Technical excellence is also a mindset, the way in which the members of an agile team conduct themselves in accordance with agile values. P3 explains: *"Technical excellence has a behavioral aspect to it. It needs the right mindset and culture."* This was echoed by P13: *"So technical excellence, it's both hard skills, so developer skills, but also, soft skills."* Behaving in accordance with agile values is expected from an individual and from an organization.

The members of an agile team are meant to embrace and exercise the agile and technical excellence mindset. P10 stated: *"Being the technical excellence is first, a mindset."* This mindset should be

exercised at every level. For example, P7 explained that it is a belief to leave the code in a "better" state after a change. He explained: *"It should be a mindset thing just to leave code better than you find it."* Our participants referred to this mindset as a collection of values and beliefs promoted by agile. P10 stated: *"That's the first thing you need to look for. A mindset attitude. It's soft. So, they are personal traits, openness, flexibility."* Having personal traits in line with agile values is highly sought in agile teams, and it *"also requires some trust in yourself may be and in others, collaboration and somehow a willingness to fail, because this approach is really trial and error, in my opinion."* (P12). The organization should also facilitate an environment where technical excellence can flourish. Sometimes it requires a cultural shift. P17 stated: *"Technical excellence is something much more than just writing good code ... [it] involves a cultural shift at the organizational level."*

In short, the term "mindset" is used by our participants to indicate a wide array of personality traits (e.g., willingness to fail), orientations towards the product (e.g., leave code better than you find it and sustainability of the design), and characteristics cultivated within teams (e.g., mutual trust, flexibility). As such, this term identifies individual attributes that are conversant with, but analytically distinct from, group attributes such as workplace culture and occupational norms.

Mindset: Technical excellence is a commitment to quality and creating sustainable software design. This commitment should emerge from the individual, as well as from the organization. It is their mutual dedication to quality and sustainable software that lead both the individual and the organization to excellence. Therefore, organizations and individuals alike should embrace excellence as a core value in their work.

Then, what is the implication of this finding? Maintaining the proper mindset necessitates continuous change in behavior for learning and growth. P5 provided an example to illustrate this point:

> *"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 ingrained in our habits. That's how we develop a mindset. If you don't have the mindset, you would not fit and would eventually leave. We had a team member who was so negative that he brought a cloud into the room every time he showed up. He was inflexible, close-minded and unwilling to learn. He didn't last. Eventually, he voluntarily left. As I said and this example shows that if you can't experiment, self-reflect as an individual and as a team, and adapt, you do not have the required mindset."*

The importance of mindset is underscored in team settings – a poor mindset not only disrupts the individual's ability to function, but also the team's work as a whole. Thus, it is important for agile teams to develop and improve their abilities through adopting a growth mindset.

Recommendation 3: To develop a mindset for technical excellence, the organization should encourage agile teams to experiment, learn and adapt.

Discussion of mindset. Change is the only constant of the universe, but to face the challenges of change a culture of excellence must be established at all levels. New mentalities and attitudes are as necessary as good practices. There must be a shift in focus, from short-term objectives toward long-range vision. A culture of excellence requires a vision of possibilities, a plan to set it in motion, and the practices to create it. Excellence is a way of being, thinking, and a commitment to activate and go beyond mediocrity. Business champions promoting business excellence succeed when they go far beyond traditional models [44].

Based in this finding, we suggest that excellence can be promoted through a mindset. The culture of excellence is people oriented and highlights the importance of empowering and actively engaging everyone. The agile vision must be communicated clearly and understood by all members of the organization. People must be purpose-based and highly determined to work together. It is not about mediocracy, but meritocracy and high performance.

Sustainable high-performing organizations create a mindset of excellence, engaging every employee with the vision, mission, and values of the organization. They also create and communicate strategies of excellence. They sustain excellence by developing strong leadership and performance skills [44]. Organizational excellence provides a set of principles and practices that can cause continuous improvement to occur; these principles and practices should be fully integrated into the regular activity of the organization [8].

4.4 Engineering practices

Engineering skills and practices are universally recognized to be key attributes of technical excellence in agile. Engineering practices is both the fourth most commonly cited attribute of technical excellence within our sample and, given that half of our interviewees did not refer to it, perhaps also the most taken-for-granted.

Engineering practices include programming language related practices and modern software engineering practices (e.g. code review and automated testing). *"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 echoed this view: *"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 make it like automate things like implementation of the CI/CD, for instance. So, things that can automate stuff rather than going manual ... Optimizing, automating, focus on quality."* Our participants discussed various software engineering practices including continuous integration, refactoring, test-driven development, automated build system, unit test framework or a practice of creating automated quality assurance tests. Engineering practices are at the core of technical excellence. Agile engineering practices are consistent with good software engineering design practices.

Recommendation 4: Solid software engineering practices are essential to technical excellence. Agile teams should adopt and continuously improve software engineering practices.

4.5 Fostering Technical Excellence

Recall that, for **RQ2**, we asked how agile teams foster "technical excellence." In response to this question, our participants experienced that technical excellence is nurtured by the following three strategies: 1) continuous learning; 2) continuous improvement; and 3) control of excellence using the "Definition of Done" (DoD).

Continuous Learning. Agile teams are meant to take actions and implement the improvement necessary to achieve efficacy and efficiency, but to do so they must first practice curiosity and learn through their mistakes. In the words of one participant, *"continuous attention to technical excellence is actually just experimenting and learning all the time. So, experiment, make mistakes and learn from the process. This is continuous attention. And share your mistakes and frustration with the team"* (P13). However, to stimulate this quality, the working environment must facilitate openness and empowerment. P10 explained: *"So in Agile, you empower your team members to own the project, to own the product, to own the deliverable, so you encourage them to take ownership, to take action, to participate not only in the execution but also in the planning, in the design. So the whole team, so that you get the best of your people, not only the execution, but also the thinking, the creativity, the commitment."*

> **Continuous learning:** Continuous learning requires organizations to encourage individuals and teams to continually increase knowledge about their behavior, own processes and practices, which is achieved by empowering them and promoting openness. New knowledge should be translated into actionable improvement initiatives.

Continuous Improvement. "In agile, you learn and improve all the time" (P9). Continuous improvement is the endeavor to continuously monitor, assess and enhance processes, methods and practices for efficiency and effectiveness. This is a core agile value. P13 asserted: *"Agile is worthless unless it serves as a catalyst for continuous improvement."* According to our participants this principle also applies to the behaviors of individuals, who are expected to invest in self-improvement. P14 explained: *"I would say this has to start from an individual level, because you would need to have people that are curious, that they want to develop, they want to learn more."* Individuals in agile teams are encouraged to be "egoless" (P2 and P9). P15 explained: *"The self-improvement for the technical excellence is that you should be a good receptor. By that I mean, you know, you should be open for the reviews that your peers are giving, your seniors are giving, because if the reviews are coming from the code or the testing or any other deliverables that you have done, if you are taking those reviews in a positive manner, then setting aside your egos, then you will be able to have a constructive feedback."*

> **Continuous improvement:** To achieve technical excellence, organizations must support software development activities by evolving their approach over time in a collaborative manner. Learning should be translated into actionable initiatives to improve the processes and the practices.

Then, what is the implication of these findings? P13 provided this example:

> *"So, 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. It is a simple process, learn, embrace, and implement. We have knowledge sharing and improvement sessions. We feel empowered to discuss our failures, shortcomings, and imperfections. Whether, it is technical, process or individuals' behaviors."*

This example shows how, when a team is empowered to act and grow, it will cultivate a habit of continuous self-improvement.

> **Recommendation 5:** Agile teams should continuously identify aspects that can be improved and develop plans of action to make these improvements.

Discussion of continuous improvement. Agile teams should be empowered to self-reflect on their processes, performance, and mindset. This reflection is necessary for teams to identify aspects that can be improved and develop plans of action to make these improvements [1].

Continuous improvement is a philosophy consisting of improvement initiatives that improve software and reduce failure, which may be translated into company-wide processes of focused improvement. The continuous improvement philosophy produces methods to enhance creativity and excellence formed within a culture of sustained improvement that reduces waste [6]. Continuous improvement describes an evolutionary learning process that builds upon a firm's path and position, with a gradual accumulation and integration of key behaviors over time through firm-specific processes [6].

Without continuous improvement activities, problem solving is random. Further, without continuous improvement activities being organized into formal structures and problem-solving initiatives, they may have minimal impact – perhaps resulting in some improvements to morale and motivation, but only in a short-lived and localized manner. Structured and systematic continuous improvement activity, meanwhile, results in sustainable improvements to project performance, with little or no bottom-line impact [5].

To realize the full benefits of continuous improvement – including cost reductions, quality improvements, and time savings – strategic continuous improvement organizational policy uses monitoring and measurement to reinforce the implementation of continuous improvement processes. A rigorous company-wide continuous improvement program promotes autonomous innovation, which has strategic benefits such as incremental problem-solving and experimentation. When continuous improvement becomes the dominant way of life in an organization, competitive advantages become evident. Everyone is actively involved in innovation processes that are both incremental and radical [5].

"Control of Excellence". Our participants indicate that excellence is controlled in every development cycle by the "Definition of Done (DoD)." The DoD is a set of criteria to determine if a deliverable is complete; these controls are opportunities to vet the artifacts for quality and excellence. DoD ensures that the product must pass quality and excellence checks before being released. 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 the 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."

> **Control of Excellence:** DoDs are measures to enforce technical excellence in every development iteration. However, DoD is only a control mechanism for excellence. Excellence is created by the people and an organizational environment that advocate for excellence.

Discussion of control of excellence. Some researchers use a multilevel approach to describe doneness, including story, sprint, release, or project. DoD uses a variety of criteria that may include software verification and validation, deploy, code inspection, test process quality, regulatory compliance, software architecture design, process management, configuration management and non-functional requirements. However, there is little agreement between done criteria in the various studies of DoD [42]. DoD is a key aspect of technical excellence, and represents a place where developers evaluate their work. However, there's no agreement on standard criteria for DoD.

> **Recommendation 6:** Agile teams should develop explicit, robust, and sustainable criteria for DoD; established criteria for DoD support technical excellence both by providing a consistent standard of quality for work (through which the organization communicates its expectations to individuals and teams) but also because once institutionalized, the DoD is itself subject to continuous improvement processes.

The recommendations of our study are a starting point to create awareness for technical excellence for organizations adopting agile. At this stage, the recommendations are derived from our data and analysis but not having an operational or concrete plans for implementations. This due to the nature of the questions we asked our participants and the scope of our research. We focused on the definition, but not on the operationalization of the Principle 9.

Not all our findings and recommendations are entirely new. Quite on the contrary, some of them have been advocated by practitioners and are known to agile researchers. Our contribution is collecting and providing empirical evidence on how **Principle 9** is interpreted and foster in practice, creating a single contribution that demonstrates agile affinity for excellence.

5 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 generalizable to non-Scrum applications.

No observation data: The research question two was investigated using evidence-based practices and not direct observations of agile teams. Direct observations may yield additional findings. The data gathered for this question reflect only how our participants believe technical excellence is nurtured, and not their actual practices.

Interviewee Transcript Review: This happens when the interviews transcripts have become available [18]. 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 for the 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 [39]. We invited all our interviewees to focus groups. However, only six participated in this exercise from the twenty invited. We arranged two focus groups, one with four and the other with two participants. We ended up with this configuration due to the participant availability. Participants were presented with our interpretations of the data and invited to comment on the findings. The participants of these focus groups were given the opportunity to either confirm or deny that the summaries of findings reflect their views, feelings and experiences. The focus group 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 focus groups transcripts available here [1].

Saturation. A common standard for conducting qualitative research [38] is saturation, that 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.

Verifiability. To allow the verifiability of our data, we made it available here [2]; to preserve the anonymity of our participants, we anonymized the interview transcripts.

6 CONCLUSION

Agile has become the de facto framework for delivering software. This popularity calls for more dissection of agile values and principles to create detailed knowledge and actionable recommendations for practitioners of agile. Technical excellence is to deliver sustainable software (future-proof design) with high-quality code (simple and clean). This is achieved by software craftsmanship and supported by a mindset. Individuals should embrace excellence as a core value. This cannot be achieved without being empowered and a working environment that operate based on openness. Organizations adopting agile should be aware that technical excellence is a culture that should be nurtured continuously.

ACKNOWLEDGMENTS

We would like to thank our interviewees and the focus group participants for their time and effort for making this study possible.

REFERENCES

[1] Pervaiz K Ahmed, Ann YE Loh, and Mohamed Zairi. 1999. Cultures for continuous improvement and learning. *Total Quality Management* 10, 4-5 (1999), 426–434.
[2] Adam Alami. 2020. The Social, Organizational and Disciplinary Aspects of Quality in Free and Open Source Software Communities. (2020).
[3] Gloria Arcos-Medina and David Mauricio. 2019. Aspects of software quality applied to the process of agile software development: a systematic literature review. *International Journal of System Assurance Engineering and Management* 10, 5 (2019), 867–897.
[4] Vebjørn Berg, Jørgen Birkeland, Anh Nguyen-Duc, Ilias O Pappas, and Letizia Jaccheri. 2020. Achieving agility and quality in product development-an empirical study of hardware startups. *Journal of Systems and Software* 167 (2020), 110599.

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

[5] John Bessant and David Francis. 1999. Developing strategic continuous improvement capability. *International journal of operations & production management* 19, 11 (1999), 1106–1119.

[6] Nadia Bhuiyan and Amit Baghel. 2005. An overview of continuous improvement: from the past to the present. *Management decision* (2005).

[7] Barry Boehm. 2002. Get ready for agile methods, with care. *Computer* 35, 1 (2002), 64–69.

[8] André M Carvalho, Paulo Sampaio, Eric Rebentisch, João Álvaro Carvalho, and Pedro Saraiva. 2019. Operational excellence, organisational culture and agility: the missing link? *Total Quality Management & Business Excellence* 30, 13-14 (2019), 1495–1514.

[9] Adnan Causevic, Daniel Sundmark, and Sasikumar Punnekkat. 2011. Factors limiting industrial adoption of test driven development: A systematic review. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. IEEE, 337–346.

[10] Jan Chong and Tom Hurlbutt. 2007. The social dynamics of pair programming. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 354–363.

[11] Alistair Cockburn. 2002. Agile software development joins the" would-be" crowd. *Cutter IT Journal* 15, 1 (2002), 6–12.

[12] IEEE Standards Coordinating Committee et al. 1990. IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990). Los Alamitos. *CA: IEEE Computer Society* 169 (1990), 132.

[13] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*. IEEE, 275–284.

[14] Torgeir Dingsøyr, Sridhar Nerur, VenuGopal Balijepally, and Nils Brede Moe. 2012. A decade of agile methodologies: Towards explaining agile software development.

[15] Tore Dybå and Torgeir Dingsøyr. 2008. Empirical studies of agile software development: A systematic review. *Information and software technology* 50, 9-10 (2008), 833–859.

[16] Martin Fowler. 2018. *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

[17] Martin Fowler, Jim Highsmith, et al. 2001. The agile manifesto. *Software Development* 9, 8 (2001), 28–35.

[18] Hadass Goldblatt, Orit Karnieli-Miller, and Melanie Neumann. 2011. Sharing qualitative research findings with participants: Study experiences of methodological and ethical dilemmas. *Patient education and counseling* 82, 3 (2011), 389–395.

[19] Jo E Hannay, Tore Dybå, Erik Arisholm, and Dag IK Sjøberg. 2009. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology* 51, 7 (2009), 1110–1122.

[20] Rashina Hoda, Norsaremah Salleh, and John Grundy. 2018. The rise and evolution of agile software development. *IEEE software* 35, 5 (2018), 58–63.

[21] Philipp Hohl, Jil Klünder, Arie van Bennekum, Ryan Lockard, James Gifford, Jürgen Münch, Michael Stupperich, and Kurt Schneider. 2018. Back to the future: origins and directions of the "Agile Manifesto"–views of the originators. *Journal of Software Engineering Research and Development* 6, 1 (2018), 1–27.

[22] Hanna Hulkko and Pekka Abrahamsson. 2005. A multiple case study on the impact of pair programming on product quality. In *Proceedings of the 27th international conference on Software engineering*. 495–504.

[23] Maarit Laanti, Jouni Similä, and Pekka Abrahamsson. 2013. Definitions of agile software development and agility. In *European Conference on Software Process Improvement*. Springer, 247–258.

[24] Thomas D LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*. 492–501.

[25] Björn Latte, Sören Henning, and Maik Wojcieszak. 2019. Clean code: On the use of practices and tools to produce maintainable code for long-living. (2019).

[26] Lech Madeyski. 2009. *Test-driven development: An empirical evaluation of agile practice*. Springer Science & Business Media.

[27] Robert C Martin. 2013. *Clean Code-Refactoring, Patterns, Testen und Techniken für sauberen Code: Deutsche Ausgabe*. MITP-Verlags GmbH & Co. KG.

[28] Robert C Martin. 2018. *Clean architecture: a craftsman's guide to software structure and design*. Prentice Hall.

[29] Pete McBreen. 2002. *Software craftsmanship: The new imperative*. Addison-Wesley Professional.

[30] Tom Mens and Tom Tourwé. 2004. A survey of software refactoring. *IEEE Transactions on software engineering* 30, 2 (2004), 126–139.

[31] Martin Michlmayr. 2007. Quality improvement in volunteer free and open source software projects–exploring the impact of release management. (2007).

[32] Raimund Moser, Pekka Abrahamsson, Witold Pedrycz, Alberto Sillitti, and Giancarlo Succi. 2007. A case study on the impact of refactoring on quality and productivity in an agile team. In *IFIP Central and East European Conference on Software Engineering Techniques*. Springer, 252–266.

[33] Nachiappan Nagappan, E Michael Maximilien, Thirumalesh Bhat, and Laurie Williams. 2008. Realizing quality improvement through test driven development: results and experiences of four industrial teams. *Empirical Software Engineering* 13, 3 (2008), 289–302.

[34] David Lorge Parnas and Mark Lawford. 2003. The role of inspection in software quality assurance. *IEEE Transactions on Software engineering* 29, 8 (2003), 674–676.

[35] Lutz Prechelt, Holger Schmeisky, and Franz Zieris. 2016. Quality experience: a grounded theory of successful agile projects without dedicated testers. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 1017–1027.

[36] Bill Pyritz. 2003. Craftsmanship versus engineering: Computer programming—An art or a science? *Bell Labs technical journal* 8, 3 (2003), 101–104.

[37] Paula Rachow, Sandra Schröder, and Matthias Riebisch. 2018. Missing clean code acceptance and support in practice-an empirical study. In *2018 25th Australasian Software Engineering Conference (ASWEC)*. IEEE, 131–140.

[38] Benjamin Saunders, Julius Sim, Tom Kingstone, Shula Baker, Jackie Waterfield, Bernadette Bartlam, Heather Burroughs, and Clare Jinks. 2018. Saturation in qualitative research: exploring its conceptualization and operationalization. *Quality & quantity* 52, 4 (2018), 1893–1907.

[39] Thomas A Schwandt. 2014. *The Sage dictionary of qualitative inquiry*. Sage publications.

[40] Robert C Seacord, Joseph Elm, Wolf Goethert, Grace A Lewis, Daniel Plakosh, John Robert, Lutz Wrage, and Mikael Lindvall. 2003. Measuring software sustainability. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings*. IEEE, 450–459.

[41] Normand Séguin, Guy Tremblay, and Houda Bagane. 2012. Agile principles as software engineering principles: An analysis. In *International Conference on Agile Software Development*. Springer, 1–15.

[42] Ana Silva, Thalles Araújo, João Nunes, Mirko Perkusich, Ednaldo Dilorenzo, Hyggo Almeida, and Angelo Perkusich. 2017. 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*. 364–373.

[43] Jamie Stevenson and Murray Wood. 2018. Recognising object-oriented software design quality: a practitioner-based questionnaire survey. *Software Quality Journal* 26, 2 (2018), 321–365.

[44] Marta-Christina Suciu. 2017. The culture of excellence. Challenges and opportunities during changing times. In *Proceedings of the International Conference on Business Excellence*, Vol. 11. Sciendo, 322–331.

[45] Olli P Timperi. 2004. An overview of quality assurance practices in agile methodologies. In *Seminar in Software Engineering*.

[46] John F Tripp and Deborah J Armstrong. 2014. Exploring the relationship between organizational adoption motives and the tailoring of agile methods. In *2014 47th Hawaii international conference on system sciences*. IEEE, 4799–4806.

[47] Colin C Venters, Caroline Jay, LMS Lau, Michael K Griffiths, Violeta Holmes, Rupert R Ward, Jim Austin, Charlie E Dibsdale, and Jie Xu. 2014. Software sustainability: The modern tower of babel. In *CEUR Workshop Proceedings*, Vol. 1216. CEUR, 7–12.

[48] Jennifer R Wolgemuth, Zeynep Erdil-Moody, Tara Opsal, Jennifer E Cross, Tanya Kaanta, Ellyn M Dickmann, and Soria Colomer. 2015. Participants' experiences of the qualitative interview: Considering the importance of research paradigms. *Qualitative research* 15, 3 (2015), 351–372.