How Scrum Adds Value to Achieving Software Quality?

Alami, Adam; Krancher, Oliver

Published in:
Journal of Empirical Software Engineering

DOI (link to publication from Publisher):
10.1007/s10664-022-10208-4

Publication date:
2022

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

Empirical Software Engineering (2022) 27:165
https://doi.org/10.1007/s10664-022-10208-4

How Scrum adds value to achieving software quality?

Adam Alami[1] · **Oliver Krancher**[2]

Accepted: 8 July 2022

Abstract

Scrum remains the most popular agile software development method implementation for a variety of reasons; one important motive is to improve software quality. Yet many organizations fail to achieve quality improvements through the use of Scrum, and existing research sheds little light on the value-add of Scrum for software quality. More specifically, (1) how notions of software quality among Scrum practitioners relate to established quality perspectives, (2) how Scrum helps teams to achieve higher software quality and (3) why some teams fail to meet the objective of higher quality. We addressed these gaps through a two-phased qualitative study based on 39 interviews and two in-depth case studies. We find that Scrum practitioners emphasize established notions of external quality comprising of conformity to business needs and absence of defects, while they also value internal quality, especially sustainable software design. Our results show that Scrum helps teams achieve both dimensions of quality by promoting some social antecedents (collaboration, psychological safety, accountability, transparency) and process-induced advantages (iterative development, formal inspection, and adaptation). Our findings unveil how these factors contribute to achieving software quality and under what conditions their effects can fail to materialize. These conditions include inconsistent Scrum implementations, cultural constraints, team tensions, and inaccessibility of end-users. In addition, the complexity of the project aggravates the impact of these conditions. Taken together, these findings show that Scrum can complement established quality assurance and software engineering practices by promoting a social environment that is conducive to creating high-quality software. Based on our findings, we provide specific recommendations for how practitioners can create such an environment.

Keywords Agile methods · Scrum · Software quality · Case studies

Communicated by: Tayana Conte

✉ Adam Alami
adal@cs.aau.dk

Oliver Krancher
olik@itu.dk

[1] Department of Computer Science, Aalborg University, Selma Lagerlöfs Vej 300, 9220 Aalborg, Denmark

[2] IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark

Published online: 20 September 2022

1 Introduction

Agile software development methods have become mainstream. While the traditional waterfall method works well when requirements are predictable, technologies are well understood, and plans are irrevocable, agile methods enable adaptation and response to change through early feedback and incremen5tal delivery (Chuang et al. 2014; Dingsøyr et al. 2012). Scrum remains the most popular agile method (digital.ai 2021), with adoption rates of around 75% (digital.ai 2021). Scrum practitioners attribute the popularity of Scrum, among other factors, to its simplicity, its ease of implementation (Schwaber and Sutherland 2017; Deemer et al. 2012) and its ability to enhance team productivity (Sutherland and Sutherland 2014). The latter claim has also been corroborated by a number of empirical studies (Layman et al. 2004; Ilieva et al. 2004; Teasley et al. 2000; Sutherland et al. 2009).

Notwithstanding these potential benefits, a key value-add of agile methods such as Scrum lies in their potential to enhance software quality. According to an agile adoption survey conducted in 2008 (Ambler 2008), 77% of the respondents claimed that agile adoption helped achieve higher software quality. Vijayasarathy and Turk (2008) survey also supported this claim (Vijayasarathy and Turk 2008). They found that the second most cited benefit of adopting agile methods was improved software quality. In a survey among Finish organizations, Rodríguez et al. (2012) found that 61% of agile adopters were motivated by improving product quality. A recent survey (digital.ai 2021) suggests that goals and benefits have stood the test of time, reporting that 42% of participating companies adopted agile to enhance software quality and that 46% were able to improve software quality (Rodríguez et al. 2012).

While these figures indicate that some teams manage to improve software quality by using Scrum, the numbers also show that the use of agile methods such as Scrum does not always result in quality improvements. This is problematic given the enormous economic costs associated with poor software quality. Software defects cost 40 to 1,000 times more to correct after the release of the software (Issac et al. 2003). The American Consortium for Information and Software Quality (CISQ) issued a report in 2018 on the state of software quality in the US (Krasner 2018). The report estimates that the cost of poor software quality in the US in 2018 was approximately $2.84 trillion. The report further states that about 60% of the software engineering effort was consumed by finding and fixing defects (Krasner 2018). Bugs in production software can have a negative impact on the business, regardless of severity. The same report claims that business disruptions caused by information technology defects can cost between $32.22 and $53.21 a minute (Krasner 2018). Software defects will continue to make their way into production. However, our efforts to prevent defects and resolve them before they are released should continue. This includes research efforts directed towards understanding how software teams can achieve higher software quality.

In sum, although poor software quality entails enormous costs and many teams use Scrum to increase software quality, these efforts are not always successful. Thus it is important to understand how teams can improve software quality by using Scrum and why these efforts sometimes fail. However, we still do not know well enough how agile methods, particularly Scrum, help achieve high software quality. While some studies suggest that agile methods help reduce defect densities (Ilieva et al. 2004; Layman et al. 2004; Li et al. 2010; Abbas et al. 2010b; Tarhan and Yilmaz 2014; Williams et al. 2011); other work did not find significant positive effects of agile methods use on quality (Li et al. 2010; Krancher 2020). Moreover, these studies do not dwell upon the way or manner in which methods such as Scrum allow teams to achieve higher quality. In addition, we lack knowledge about

what agile methods do well and do differently to facilitate achieving software quality. Key principles of the manifesto (Beck et al. 2001), e.g., "individuals and interactions over processes and tools", "customer collaboration over contract negotiation" and "responding to change over following a plan" may indicate that the value-add of agile methods is strongly related to values rather than specific engineering practices. How this shift from a process-heavy approach to a value-based approach enhances the ability of software development teams to achieve quality remains poorly understood. In addition, Arcos-Medina and Mauricio (2019) literature review shows that most factors identified to improve quality in agile teams are a combination of non-engineering (e.g., "self-organizing" and continuous learning) and engineering practices (Arcos-Medina and Mauricio 2019). Notably, "teamwork" and "management" practices were often cited in the literature to advance quality (Arcos-Medina and Mauricio 2019). Furthermore, even though the studies identified above suggest a potential positive effect of the use of agile methods on quality, the latter is not explicitly mentioned in the Agile Manifesto. This suggests that much remains to be learned about the relationship between agile methods use and quality. In light of these gaps, our study examines three research questions. The first focuses on the concept of software quality, which is rather complex "because it means different things to different people" (Kitchenham and Pfleeger 1996). Given that definitions may evolve over time, it is sound to revisit them, especially after a period where new methods have taken over and new ways of developing software have been advocated and adopted. It is also an opportunity to assess whether a consensus has emerged about the meaning of software quality. We want to understand agile practitioners' perspectives on the definition of software quality, which also allows us to discuss our findings on our subsequent research questions, with a more relatable definition of software quality. Hence, we investigate:

RQ1: How Do Scrum Practitioners Define Software Quality?

Our second research question focuses on how Scrum teams achieve higher software quality. Traditional approaches to software development assure quality by having quality gates at the transitions between project phases and by extensive tests at the end of development. Traditional approaches also emphasize good software engineering and quality assurance practices (e.g., continuous integration, coding standards, peer review and automation), which have been advocated for decades to achieve quality. In the Agile Manifesto, quality is an attendant goal, not explicitly advocated. Moreover, there is a paucity of empirical research that shows how agile practitioners achieve higher software quality using Scrum. Thus, it remains unclear how agile teams' endeavors to attain higher software quality compare to traditional approaches. Eliciting knowledge on how software quality is achieved in the context of agile software development is relevant because it would make this knowledge explicit in the software engineering literature, presenting thus an opportunity to reduce the theory-practice gap, i.e., our theoretical understanding may not be universal in the industry or discipline-wide. To recap, software quality control and assurance practices have existed for decades prior to agile. The agile proposition is to influence values, principles, events and activities in software development. The focus of our second research question lies on how this proposition helps teams to achieve software quality:

RQ2: How Do Scrum Values, Principles and Prescribed Events and Activities Advance Achieving Software Quality?

Our third research question focuses on why Scrum teams may fail to achieve higher software quality. Although surveys have consistently reported promising results from agile

adopters, they also tell a hidden story. For example, the latest available survey (digital.ai 2021) reports that 46% of agile adopters have achieved enhanced software quality. Then, what happened in the other 54% of the surveyed companies? That is, why did some companies not achieve improvements in the quality of their software? Even in the most enthusiastic survey, where 77% of the respondents reported increases of software quality, 23% of the surveyed participants failed to enhance software quality. The literature cited above focuses primarily on positive cases, i.e., noticeable improvements in software quality. We believe it is essential to understand and report examples of the stories of the 54% (in digital.ai 2021 survey) and 23% (Ambler 2008 survey) cases. Hence, we propose to investigate:

> ***RQ3:*** What Are the Circumstances and the Conditions which Impede Scrum Teams from Improving the Quality of their Software?

We investigated our research questions using a two-phased qualitative research design based on interviews and case studies. In the first phase, we conducted interviews to capture the experience and perspectives of Scrum practitioners, addressing **RQ1** and **RQ2**. To do so, we recruited practitioners with extensive experience in software development, managing software teams, software developers, quality assurance practitioners, and participants with responsibilities overseeing the delivery of software products using Scrum. It is particularly relevant to understand how practitioners have managed to successfully use Scrum to achieve quality. Not only does this deepen our understanding of how practitioners use Scrum to achieve quality; it also provides insights for those who pursue using this method. Whereas Phase 1 focused on positive experiences using Scrum (i.e., practitioners with experiences achieving better software quality using Scrum), Phase 2 zoomed into two cases with limited improvements in software quality after more than two years of its implementation. Thus, Phase 2 addressed **RQ3** through a study of two negative cases. This juxtaposing of positive (**RQ2**) and negative (**RQ3**) experiences provided a balanced perspective and was instrumental in arriving at a synthesis of our conclusions. We found:

- **RQ1:** Even though our participants were somehow reluctant to define software quality, stating the subjectivity of the concept, most participants emphasized properties that we subsume under *external quality*, including conformity with business needs and absence of defects. These properties are in line with well-established industry standards (e.g., ISO/IEC 25010 ISO/IEC 2011). Some participants, especially those recruited from positive cases, also emphasize properties related to *internal quality*, including a sustainable software design that supports the continuity of the software and eases teamwork.
- **RQ2:** We identified the added value of Scrum for achieving software quality. It is a combination of social qualities and process-induced advantages that strengthen the team's ability to deliver software within the quality expectations. While quality control (e.g., software testing) and assurance (e.g., code review) techniques are prevention-oriented and detection-oriented, respectively, the value added by Scrum lies in social foundations and process-induced advantages that advance the team's ability to deliver software quality. For example, Scrum values promote psychological safety in the development environment, which instills developers to feel "safe" to invest effort in assuring quality, e.g., writing elegant and readable code. Scrum principles and values also promote collaboration, which facilitates knowledge sharing within the team, resulting in a better understanding of the requirements and fewer assumptions that may result in defects.
- **RQ3:** We found that the potential value-add of Scrum for achieving software quality can be compromised by constraints. These constraints can prevent the value-add

of Scrum by inhibiting social antecedents or process-induced advantages that are promoted by Scrum. For example, in the cases we studied, we found that cultural constraints and team tensions restricted their Scrum implementations to produce some of the results reported by Phase 1 participants. Cultural constraints hampered, for example, the development of psychological safety, leading to an environment where engineers feared to speak up. In the second case, tensions emerged in the team due to clashes in expectations and mismatches of styles of work. While the team leadership exercised control over the team and displayed an expectation of "no bad news, only good news," the team members wanted more leeway to influence and safety to raise their concerns. These conditions rendered collaboration inefficient, and the lack of a psychologically safe environment resulted in disengaged and less caring developers.

We start by reviewing relevant literature in Section 2. Then we describe our methods in Section 3. Section 4 is devoted to the analysis and interpretation of the data. We validated our findings and report the results in Section 5. We discuss the implications of our findings in Section 6 and make recommendations for organizations seeking to implement Scrum to elevate their teams' abilities to achieve software quality. We highlight the limitations and assumptions of the study in Section 7. Section 8 is devoted to discussing threats to validity. We conclude in Section 9.

2 Related Work

We reviewed the literature to relate our study to the ongoing work in software engineering, identify gaps, and provide a benchmark for comparing our results with other findings (Creswell and Poth 2016). In line with this goal and with traditions in qualitative research (e.g., Alami and Paasivaara 2021; Alami et al. 2022; Prechelt et al. 2016), the focus of this literature review was selective, in contrast focus systematic literature reviews, which aim at systematically identifying, analyzing, and interpreting all available evidence on a topic (Kitchenham et al. 2009).

We used combinations of keywords to identify related work. We searched several research databases including Scopus, Web of Science, IEEE Xplore, ScienceDirect and Google Scholar. We used combinations of these keywords in our search, "software quality", "quality assurance", "quality", "agile", "agile methods", "Scrum", "extreme programing", and "XP." We assessed the relevance of the identified work by reading the title, then the abstract. In some cases, that was not sufficient to determine the relevance of the work. When that was the case, we read the findings section to inform our decisions.

The question of whether Scrum adoption has significance in achieving software quality has attracted some interest in both the information systems and the software engineering community. Most of the available work investigates this question by assessing changes in product quality before and after Scrum implementation (e.g., Abbas et al. 2010a; Li et al. 2010) or by comparing plan-driven and agile teams (e.g., Tarhan and Yilmaz 2014; Williams et al. 2011; Layman et al. 2004). Using the number of defects as an indicator of software quality, these studies show that Scrum and other agile methods can help teams to achieve higher software quality (e.g., Abbas et al. 2010a; Li et al. 2010; Layman et al. 2004). Still, with few exceptions (e.g. Prechelt et al. 2016), the available literature has not examined in what way or manner or by what means Scrum makes a difference.

Throughout the past decades, research on the relationship between agile methods and software quality has focused on comparison studies (Tarhan and Yilmaz 2014; Layman

et al. 2004; Williams et al. 2011), the impact of the adoption of an agile method on software quality (Li et al. 2010; Abbas et al. 2010a; Green 2011), in-depth examination of how software team(s) achieve software quality using agile methods (Prechelt et al. 2016) and the effect of geographical dispersion on quality in agile teams. Methodologically, most of these studies opted for case studies to carry out their research, with few instances of survey research (Abbas et al. 2010a; Krancher 2020) and research relying on survey and archival data (Green 2011). The preference for case studies is understandable as this method provides access to a particular setting with the purpose of examining it in depth to enrich our understanding of the problem being investigated (Yin 2018). Below we present and discuss the related work chronologically. At the end of the section, we summarize the outcome of this review in Table 1 and highlight the delta with our work.

2.1 Early 2000's

This period saw the emergence of early work on the topic, especially (Layman et al. 2004) work. They conducted a longitudinal case study to examine the impact of Extreme Programming (XP) practices on software quality. They evaluated and compared two releases of the product, where the first used XP and the second did not (Layman et al. 2004). For the XP release, the defect rate was reduced by 65% pre-release, and 35% post-release, well below the industry average ((Layman et al. 2004). Although the team in Layman et al.'s study used XP and not Scrum, XP and Scrum are both based on agile principles, suggesting that parallels between teams using the two methods can be drawn. Even though not directly related to software quality, Hanssen and Fægri (2006) report a case study of a team transitioning to agile methods. They found that enhanced customer participation yielded benefits such as improved understanding of users' problems, which may have benefited software quality.

Sutherland et al. (2009) investigated a distributed software development team using Scrum to assess its performance and defect rates. The team was co-located in the Netherlands prior to the company making the decision to relocate part of the team's activities to India. The study used data reported prior to the team becoming distributed to compare defect rates pre and post relocation. For productivity, they used previously reported data (Cohn 2004) of co-located Scrum teams as a point of reference.

After the team became distributed, open defects during the course of the project remained unchanged, and the number of open defects per KLOC has decreased. In addition, 90% of the defects found were resolved in the same iteration in which they were found or introduced (Sutherland et al. 2009). For productivity, they used the number of lines of code, function points (FP), and FP per month. Compared to Cohn's study (2004), even distributed, team productivity matches co-located Scrum teams (Sutherland et al. 2009). The point made by this study is that a rigorous implementation of Scrum reduces barriers inherent to the nature of distributed teams. Scrum enhanced communication, coordination and helped reduce the impact of cultural differences. These social qualities helped the team to maintain its productivity level and the quality of the software they delivered (Sutherland et al. 2009). Our study further unfolds how Scrum influences a team's ability to achieve software quality.

Even though this early work did not make strong conclusions about the impact of agile methods on software quality, it inspired subsequent work. After 2010, new work (e.g., Williams et al. 2011) continued in the same line, i.e., comparing the effect of agile methods implementation on software quality to plan-driven approaches. In addition, researchers (e.g., Abbas et al. 2010a; Li et al. 2010; Green 2011) during this period, became interested in evaluating the improvement in software quality before and after the adoption of agile methods.

Table 1 Related work and the differences to our work

Period	Related work	Differences to this study
Early 2000's	Layman et al. (2004), Hanssen and Fægri (2006), Sutherland et al. (2009), Cohn (2004)	These works are concerned with the outcome (i.e., does the software quality improve?) and not in what way or manner agile methods help. Our work sheds some light on how Scrum adds value to teams pursuit for better quality. We conclude that Scrum brings about social qualities, such as collaboration and transparency, to the team. These qualities promote behaviors such as knowledge sharing and voluntary inspections which have subsequent effects on improving quality.
After 2010	Abbas et al. (2010a), Li et al. (2010), Green (2011), Williams et al. (2011), Hoda et al. (2011)	These works evaluate quality before and after the adoption of agile methods. These studies concur with the work cited above, i.e., agile methods can make a difference when it comes to software quality. Still, examining only the outcome (i.e., does the software quality improve?) has inherent limitations; we still do not know what agile methods bring into the table to create the effect of improving quality and in particular why some teams fail to achieve similar outcome. In addition to showing how Scrum adds value to achieving software quality, our work also identified potential constraints that hinder teams ability to do so.
After 2014	Tarhan and Yilmaz (2014), Prechelt et al. (2016)	These work highlight the importance of frequent feedback, including from the end users in order to improve software quality in agile methods. These works have parallels to ours. Our work builds on these studies and shows additional social and process enablers promoted by Scrum which help advancing teams' quest for better quality. Some of our findings (e.g., psychological safety, accountability and transparency) have not been reported previously to influence software quality.
After 2018	Krancher (2020)	Software quality was not the focus on the study. The study shows that different agile practices may have different effects on software quality and that the effects can depend on requirements risk. Thus, these works highlights the need to understand in greater depth how and why agile methods impact software quality and how these effects depend on project conditions. Our work addresses these two issues through RQ2 and RQ3.

2.2 After 2010

During this period, three studies (Abbas et al. 2010a; Li et al. 2010; Green 2011) looked at the impact of agile methods adoption on software quality. While Abbas et al. (2010a) and Green (2011) reported a noticeable improvement in quality, Li et al.'s (2010) conclusions are not entirely aligned with the former. They suggested that Scrum influenced the efficiency of the team, including their quality assurance processes, but had no "significant" impact on defect density.

Abbas et al. (2010a) conducted a survey study to investigate the impact of "agile projects governance" on quality. Their overall conclusion is that software quality improves with the use of agile methods. Organizations that achieved higher software quality sought frequent customer feedback and had more efficient and impactful retrospectives (Abbas et al. 2010a). They also found a positive correlation between agile experience in the organization and code quality, 67% of the respondents reported experiencing high code quality. Organizations showing more mature agile implementation had a slightly higher rate of 79% (Abbas et al. 2010a).

Green (2011) investigated the results of Scrum adoption at Adobe Systems using a survey and historical defects data. The author investigated nine teams with available historical defects data. Seven of the nine teams experienced improvements in defect rates after the transition to Scrum. One particular team did not follow the pattern observed in other teams. This team also reported low satisfaction with their Scrum implementation. The author concludes that improvements in defect rates are highly correlated with teams' satisfaction with their Scrum implementations (Green 2011).

Li et al. (2010) conducted a longitudinal case study of a software development team to investigate changes in software quality before and after the transition to Scrum. The study concluded that after the transition to Scrum, the team did not experience "a significant reduction of defect densities"; however, there was an improvement in the efficiency of quality assurance processes used by the team. They found that applying Scrum ensured that defects were dealt with within reasonable timeframes. Early testing reduced code defects by 42%. As a consequence, the team felt that Scrum reduced the amount of time wasted. They attributed the enhancement in quality assurance efficiency to some Scrum practices (e.g., short sprints) that mitigated the risk of not resolving issues promptly and to the daily Scrum meetings, which allowed immediate feedback (Li et al. 2010).

Williams et al. (2011) compared three software teams at Microsoft, investigating whether the teams managed to improve the quality of their products using Scrum. They used a previous study (Maximilien and Williams 2003) as a reference point for their comparison. This study also concluded that agile methods can help teams to improve the quality of their products.

Hoda et al. (2011) performed a grounded-theory study in 16 organizations that relied on self-organizing teams using agile methods. Although software quality was not the focus of their study, they found that difficulties in mobilizing sufficient customer involvement led to delayed feedback and rework (Hoda et al. 2011), which surfaced in defects and, hence, in software quality issues.

Noticeably, these studies, including early 2000s, provide evidence that the adoption of agile methods can elevate a team's ability to achieve quality and that feedback processes play an important role in this realm. Although in both periods researchers used different approaches, comparing agile to plan-driven projects and evaluating quality before and after the adoption respectively, they have a similar limitation. They provide limited insights into the question of how and why some teams achieve higher quality with agile methods while others do not. Practitioners would benefit from a more comprehensive understanding of how they can leverage Scrum to achieve results similar to those reported in successful cases.

2.3 After 2014

After 2014, researchers continued to show interest in comparative studies (e.g., Tarhan and Yilmaz 2014), but more importantly, we found work (i.e., Prechelt et al. 2016) that has explicitly examined how quality is assured in agile teams.

Tarhan and Yilmaz (2014) conducted a multiple-case study of two similar software development projects. The study's purpose was to compare and contrast the plan-driven method with agile in regards to process performance and product quality. The defect density in the system test phase was 14% higher in the plan-driven than in the agile development project. In the customer testing phase, the defect density was five times higher than in the agile project (Tarhan and Yilmaz 2014).

Prechelt et al. (2016) performed an exploratory holistic, multiple-case study to explicitly examine how quality was assured in three agile project teams. They focused on teams that did not have a dedicated tester role and on the advantages and disadvantages of not using testers. All three teams worked on the same domain, which was in-house development of a single, large web portal. All three teams had similarities as follows: millions of users, different user types, individual customers accounting for only a small part of the revenue stream, and complex functionality, scalability, and access channels (Prechelt et al. 2016). Prechelt et al. (2016) propose "Quality Experience", a mode of quality assurance and deployment where each team member feels fully responsible for the quality of their software and receives feedback about its quality. Such feedback is quick, realistic, and contributes to the repair of deficiencies. They further report that empowerment was associated with increased responsibility, especially the responsibility for building automated tests. Developers also participated in the requirements definition based on their increased understanding of the domain from increased feedback from the field and high motivation. Quick, direct and realistic feedback enabled the practice and capability of "Rapid Repair" (Prechelt et al. 2016). Frequent deployment was also a benefit of agile practices, even as often as several times a week. Frequent deployments made not having separate testers bearable because failures were short-lived. Debugging was simpler because changes were smaller, realistic feedback was more available, and better strategies emerged for achieving larger goals (Prechelt et al. 2016).

In summary, Prechet et al.'s study emphasizes the importance of frequent feedback, including feedback from customers, as one important mechanism for achieving quality in agile methods. Moreover, the study shows that modular architecture, comprehensive test automation, and deployment automation can help ensure quality even without designated testers (2016). Especially Prechelt et al.'s study has considerable synergy with ours. Some of the agile influencing factors reported in this study, e.g., feedback, and "feeling responsible" (2016), appear in our conclusions. We build on some of these findings by exploring additional social and process-related benefits, how these benefits materialize in teams, and how they affect quality. We build on this work by combining data from practitioners and case studies with testers with data from teams not using testers. Moreover, extending Prechelt et al.'s work, we also focus on how and why the social processes for quality may break down in certain cases and why some Scrum teams do not achieve results similar to what was reported by the first and second streams discussed above.

2.4 Recent Work—After 2018

Arcos-Medina and Mauricio (2019) conducted a systematic review of the agile literature on quality. The review shows that software quality is influenced by various factors in agile software development (Arcos-Medina and Mauricio 2019). They identified five critical success factors, teamwork practices, engineering practices, management practices, documentation practices, and testing practices. However, as for most studies reported in this review, quality was not the primary topic of investigation (Arcos-Medina and Mauricio 2019). This

literature study shows that more empirical work is needed to investigate how agile methods assist and influence teams' ability to attain better software quality.

Jain et al. (2018) analyzed the impact of agile, as an approach to software development, on software quality. They mapped several agile practices to software quality attributes. This mapping shows how agile practices can contribute to fulfilling some software quality attributes (e.g., reliability and maintainability). They suggest that Scrum Sprints contribute to improving software reliability and maintainability. Presenting business requirements in a User Story format allows effective reviews and subsequently improvement in reducing functionality defects and performance (Jain et al. 2018). However, this study is not empirical; the conclusions are based on the authors' assumptions and hypothesizing.

Krancher (2020) conducted a survey study among sponsors and developers in 60 outsourced software projects to examine how the use of specific agile practices correlates with quality (or effectiveness in his terms). He found insignificant effects of continuous integration and joint decision-making and a significant negative effect of continuous analysis on quality. He also found that the impact of agile practices on quality depended requirements risk such that continuous integration was more beneficial and continuous analysis less detrimental in projects that had more uncertain requirements. Although this study shows that different agile practices can have different effects on quality, it does not answer the question of how (i.e., through which causal mechanisms) these agile practices affect quality. Moreover, while it highlights that circumstances and conditions, such as requirements risk, may affect how agile methods affect quality, it focuses on one condition only, leaving upon the question of what other conditions may impede or enable teams to increase software quality using agile methods (i.e., our RQ3) (Krancher 2020).

Table 1 summarizes the related work on software quality and discusses the variations from our work. The current work supports the claim that the adoption of Scrum (or other agile methods) enhances a team's ability to achieve software quality. However, we know little about how Scrum adds value to the quest for quality. In order to support teams in their efforts to better capitalize on the strengths of Scrum to elevate their capabilities to achieve quality, we propose to decipher how Scrum values, principles, process activities, and events help in the pursuit of quality. Our work examined extensively how 39 Scrum practitioners experienced enhanced software quality and why two cases failed to do so.

3 Methods

Figure 1 depicts the research process we carried out in this study. This study is a mixed-method study implemented in two phases. In the first phase, we focused on the successful implementation of Scrum (**RQ2**). We recruited practitioners who experienced good results in achieving software quality using Scrum. We relied on semi-structured interviews to capture practitioners' experiences. In the second phase of the study, we selected two companies that use Scrum. For some reason, both companies were not successful in capitalizing on the method to improve the quality of the software they were producing (**RQ3**). Phase two aimed to identify the variance in unsuccessful cases, which helped us further understand the patterns and trends observed in the first phase. Patton (1999) suggests that the understanding of the phenomena being studied expands when negative cases are considered. Including data from cases that contradict previously drawn conclusions strengthens the qualitative inquiry's validity and enriches the results (Hanson 2017). Furthermore, combining positive and negative cases prevents survival bias; relying only on positive cases may generate

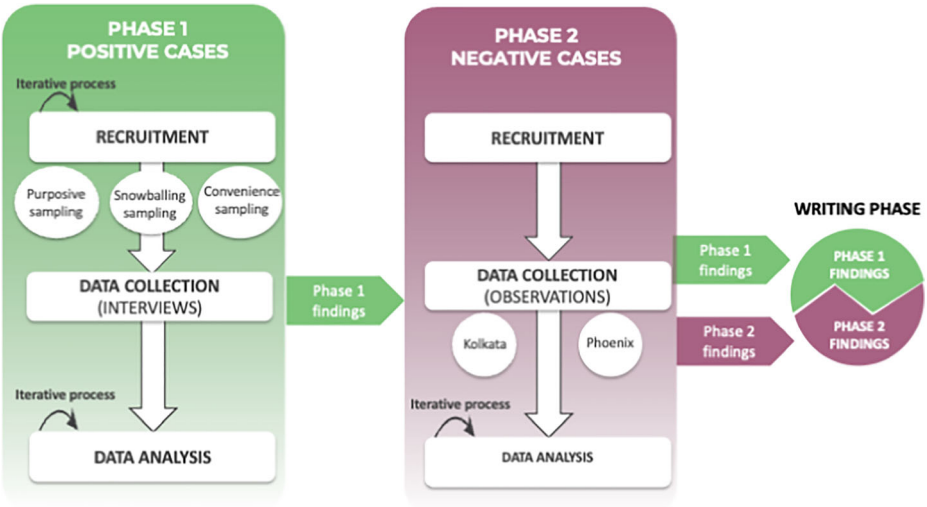

Fig. 1 Research process

overly optimistic conclusions. We used data from both phases to answer **RQ1**. While the first phase provided us with a working definition, in the second phase we synced the definition with perceptions of software quality in the two cases. As shown in Fig. 1, Phase 1 findings served as an analytical tool to understand and explain Phase 2 cases. We synthesized and presented the findings of both phases as part of the writing process. This allowed us to tell a cohesive and integrated narrative.

3.1 Phase 1 (RQ1 & RQ2)

We recruited Scrum practitioners with experience using Scrum. Knowledge creation using informants' or practitioners' experiences is widely used in all sciences, and it is a valid source of data. We sought to understand the practitioners' experiences regarding how they managed to use Scrum to elevate their teams' ability to achieve software quality. We used information and knowledge held by Scrum practitioners who successfully capitalized on the method and its values and principles to achieve quality. Practitioners have an emic perspective; they have insider information or knowledge about Scrum implementation and its ability to achieve quality. This knowledge is difficult to obtain otherwise. In addition, using this type of knowledge means reducing the tension between theory and practice.

3.1.1 Participants Characteristics

There are limitations when relying on one particular type of participant (e.g., developer only) and not the others. We could end up with conclusions that are one-sided or biased. To counter this drawback, we preferred using different roles to achieve a complete view. We aimed for a sample that favored subjects at the front line of influencing the end product quality, i.e., software developers and quality assurance analysts (QA). These roles made up 56% of our sample. We used five different roles to shape our sample: management, product owners, Scrum Masters, quality assurance analysts and software developers. Below, we discuss the relevance of these roles to our study.

- **Management:** Managers evaluate the organization's investment in their resources, processes, and methods. The investment in Scrum is meant to achieve productivity and enhanced software quality. The management perspective is relevant to assess whether their goals have been met.
- **Product Owners:** The Product Owner (PO) represents the needs and expectations of business stakeholders. Software quality can significantly impact end-user satisfaction, cost of ownership, and software life expectancy. In order to align the stakeholders' expectations on quality, the Product Owner continuously communicates those expectations to the team. This perspective is vital because product owners have intimate encounters regarding whether the expectations of their stakeholders on quality are met and influenced by Scrum implementation.
- **Scrum Masters:** The Scrum Master's role is to ensure that Scrum practices are adhered to by the team. They also have the "bird's eye view" of the process and the output. They have intimate knowledge to know if the process has any impact on achieving better software quality.
- **Quality Assurance Analysts:** The responsibility of quality assurance analysts (QAs) is to apply quality assurance practices and principles throughout the software development process. They are the gatekeepers of quality. They assess whether the software meets the expectations on quality. Their perspective is relevant because they have first-hand experience regarding the ability of Scrum to enable achieving better software quality. Although some studies (e.g., Prechelt et al. 2016) reported that some Scrum teams do not use QAs. However, in our search for participants, we encountered a significant number of QAs as part of Scrum teams. In addition, one of the Phase 2 cases used QAs to control and assure quality expectations. Our search for participants and cases for Phase 2 showed that the QA role remains relevant and important to most agile teams, even though some choose to carry out quality assurance activities without QAs. Disregarding this role in our sample may lead to conclusions skewed toward developers and other non-development roles' claims.
- **Software developers:** This is an obvious choice. Software developers write the code and produce the structure or internal workings of the software. They have the "nuts and bolts" knowledge to know how Scrum helps them to achieve quality.

3.1.2 Sampling

We used three sampling techniques for selecting the participants for Phase 1: (1) purposive sampling, (2) snowballing and (3) convenience sampling.

Purposive Sampling Purposive sampling is the deliberate choice of an informant due to the qualities which the informant possesses (Patton 2014). This non-probability sampling technique is most effective when the researcher needs to study a particular phenomenon using experts (Patton 2014). The inherent bias of the method contributes to its efficiency. The criteria used by the researcher to select potential participants lead to more proficient and well-informed participants (Patton 2014). We defined the qualities which practitioners should have, keeping in mind the competency of our participants in our selection. Table 2 describes the criteria that we set and explains why we used them. These criteria have been used for all our sampling techniques of Phase 1. We used LinkedIn to find potential participants by using the "search people" feature to search for Scrum practitioners. We used various combinations of these keywords: "Scrum", "Agile", "Software Developer", "Scrum

Table 2 Selection criteria used to identify and recruit phase 1 participants

Selection criteria	How we used the selection criteria in our recruitment process	
Five years minimum experience working part of a Scrum team or overseeing a software delivery function that uses Scrum (e.g., CTO or Program Manager)	Purposive sampling	We used the potential participant's LinkedIn profile to validate this requirement before we sent the invite to participate. After we established contact, we sent pre-selection questions (see Table 3) to further assess their suitability for the study, we used similar process for all sampling methods.
	Convenience sampling	Once a potential participant was recommended by our contacts in the industry, we sent an email with questions to assess his/her suitability for the study. We also asked for a copy of the resume or the link to their LinkedIn profile to confirm.
The participant has been part of a Scrum team successfully that has helped to produce quality software	Snowball sampling	While participants recruited using convenience sampling were recommended by our industry contacts, we used snowballing to recruit additional participants by asking participants already interviewed to indicate further potential participants. Once a potential participant was recommended, we made a preliminary contact via email and sent questions to assess their suitability for the study. We also asked for a copy of the resume or the link to their LinkedIn profile to confirm.
	All sampling techniques	After we established contact with a potential participant, we asked via email or LinkedIn chat about whether Scrum has helped their teams to achieve software quality. Only respondents who positively confirmed and praised Scrum for helping achieve quality were asked to participate (see Table 3 for pre-selection questions).

Master", "Product Owner", "Quality Assurance", "Program Manager", "Project Manager", "CTO", "Head of IT", and "Software Manager". We relied on the "About" section of profiles to assess the participant's suitability. For example, participant 8 has in his "About" section: "track record of driving Lean/Agile improvements at scale across product and technology." Some of these search combinations resulted in over 5 million profiles. We sent "InMail" invites to 213 potential participants over three months. After further selection (see Table 3), we recruited 27 participants.

Snowballing We used snowballing to recruit further participants. After the interviews, we sent an email to thank the participant for taking part in our study. In the email, we asked whether they could refer us to another participant. We received 13 contacts from our participants (already interviewed). We contacted all 13 and successfully recruited three additional participants.

Table 3 Phase 1 participants selection questions

#	Pre-selection questions sent via email or LinkedIn InMail to potential participants
SQ1	How many years of experience do you have in software development?
SQ2	How many years of experience do you have working in Scrum team, implementing Scrum or overseeing a delivery capability with Scrum teams?
SQ3	Do you think Scrum helped your team or organization to improve the quality of their software?
SQ4	Can you, briefly, explain how does Scrum help your team or organization to improve software quality?

Convenience Sampling Both methods described above yielded only five software developers in the sample. Given that this role plays a crucial role in shaping the quality of the software, we used convenience sampling to recruit additional participants. We used our industry contacts to refer us to software developers and received 12 referrals. We successfully interviewed nine participants after the selection process.

Prior to recruiting a participant for Phase 1, we sent either an email or a LinkedIn InMail with the questions listed in Table 3. The answers to these questions allowed us to assess and validate their suitability to participate in the first phase of the study. This process was used for all sampling techniques (Tables 2 and 4).

Table 5 summarizes the characteristics of our sample. We used the role as labeled in their LinkedIn profile or resume. Experience indicates the total number of years the participant spent working in the software development industry (i.e., column Soft. Dev.) and the number of years in a Scrum environment (i.e., Scrum column). Affiliation describes the organization in which our participants work. The type of business and the number of employees (i.e., No

Table 4 Key parts of the interview's questions

Introduction questions
Can you please introduce yourself and talk about your experience?
What do you think of agile? What is your opinion of it?
How do you define "software quality" in the context of agile software development?
Core questions
Can you describe your Scrum environment? For example, how do you work as a team? How do you use Scrum?
Do you think this Scrum implementation is working for your team and why?
What do you do to assure software quality in this Scrum process?
Do you think this Scrum implementation produces quality software and how?
How does, for example, your Scrum implementation help to find bugs?
How does, for example, your Scrum implementation help to produce high quality code? How does your Scrum environment motivate you to achieve code quality? (For software developers only) Can you take me through the journey of a requirement/feature/user story in your Scrum implementation and explain how you ensure quality throughout the journey?
Probing questions
Can you share with me an example when working on Scrum motivated you to write better code?
Can you share with me an example from your experience on how Scrum helped achieving software quality?

Table 5 The study sample

#	Sampling	Role	Experience		Education	Affiliation		Country
			Soft. Dev.	Scrum		Type of business	No. Emp.	
Management								
P1	Purposive	Senior Project Manager	7	7	Bachelor's in project management	Information Technology	630	Belgium
P2	Purposive	Head of IT	17	10	Master's in computer science	Information Technology	537	UK
P3	Purposive	Senior Program Manager	8	6	Bachelor's in computer science	Information Technology	205,857	USA
P4	Purposive	Program Manager	13	8	Bachelor's in electrical engineering	Hospital & Health Care	38,233	India
P5	Purposive	Program Manager	8	8	Master's in computer science	Technology Startup	117	USA
P6	Purposive	Software Delivery Manager	10	10	Bachelor's in software engineering	Financial Services	233	USA
P7	Purposive	Program Manager	20	11	PhD in computer science	Information Technology	110	Turkey
P8	Purposive	Software Delivery Manager	7	7	Bachelor's in computer science	Biotechnology	292	Serbia
Product Owner								
P9	Snowballing	Product Owner	10	7	Bachelor's in computer science	Real Estate	153	India
P10	Snowballing	Product Owner	15	12	Master's in project management	Financial Services	10,955	Russia
P11	Purposive	Product Owner	6	6	Master's in computer science	Airlines	51,601	Canada
Scrum Master								
P12	Purposive	Scrum Master	6	6	Bachelor's in computer science	Information Technology	1,504	UK
P13	Purposive	Scrum Master	5	5	Bachelor's in software engineering	Sporting Goods Manufacturer	41,276	UK
P14	Purposive	Scrum Master	19	10	Bachelor's in computer science	Information Technology	1,076	Ireland
P15	Purposive	Scrum Master	12	7	Bachelor's in information technology	Capital & Private Equity	504	USA
P16	Purposive	Scrum Master	14	9	Bachelor's in computer science	Information Technology	471,512	UK
P17	Purposive	Scrum Master	8	8	Master's in computer science	Banking	5,443	Netherlands
Quality Assurance								
P18	Purposive	QA Team Lead	6	6	Master's in telecommunications engineering	Online Media Services	431	Serbia
P19	Purposive	QA Engineer	8	8	Bachelor's in computer science	Information Technology	114	UK

Table 5 (continued)

#	Sampling	Role	Experience		Education	Affiliation		Country
			Soft. Dev.	Scrum		Type of business	No. Emp.	
P20	Purposive	QA Engineer	7	7	Master's in data science	Non-profit Organization	338	Australia
P21	Purposive	QA Engineer	9	8	Bachelor's in informatics	Games Development	3,939	Australia
P22	Purposive	QA Engineer	11	9	Bachelor's in electronics	Information Technology	532,421	Germany
P23	Purposive	QA Team Lead	11	10	Bachelor's software engineering	Information Technology	487	USA
P24	Purposive	QA Team Lead	6	6	Bachelor's in electrical engineering	Automotive Manufacturer	4,654	Sweden
P25	Purposive	QA Team Lead	11	9	Bachelor's in communications engineering	Information Technology	877	India
Software Development								
P26	Purposive	Senior Node.js Developer	16	12	Bachelor's in computer science	Technology Startup	134	Poland
P27	Purposive	Senior Software Developer	20	10	Bachelor's in computer science	Biotechnology	247	USA
P28	Purposive	Software Developer	5	5	Master's in computer science	Industrial Engineering	982	Italy
P29	Purposive	Senior Software Developer	8	8	Bachelor's in software engineering	Financial Services	14,157	USA
P30	Convenience	Software Engineer	7	7	Bachelor's in computer science	Technology Startup	26	UK
P31	Convenience	Senior Software Developer	17	12	Master's in software development	Information Technology	161	Denmark
P32	Convenience	Senior Software Developer	11	11	Bachelor's in computer science	Information Technology	543	UK
P33	Convenience	Senior Software Engineer	30	10	Bachelor's in computer science	Financial Services	52,227	UK
P34	Convenience	Software Engineer	5	5	Bachelor's in electronics engineering	Information Technology	5,934	India
P35	Convenience	Software Engineer	5	5	Master's in computer science	Electronic Manufacturing	8,192	UK
P36	Convenience	Senior Software Engineer	17	12	Master's in computer science	Information Technology	75,355	UK
P37	Convenience	Senior Software Engineer	17	9	Bachelor's in software engineering	Technology Startup	119	Australia
P38	Convenience	BackEnd Developer	15	12	Bachelor's in computer science	Education	3,540	UK
P39	Snowballing	Senior Software Engineer	13	10	Bachelor's in computer science	Information Technology	471,512	India

Emp) were taken from the company's websites or LinkedIn profiles. The country column is where the participant is located and works.

3.1.3 Data Collection

We used semi-structured interviews to collect the data for this phase of the study (see evidence from the field, Fig. 5 in Appendix 1). This form of interview is flexible, allowing new questions to be brought up during the discussion in addition to a predefined interview guide. This flexibility allowed us to enhance the depth of the conversations. We structured the guide into three categories: introductory, core, and probing questions, see Table 4. The introductory questions were for starting the conversation and building up to the core topic of the interview. The core questions were aimed at gathering data related to our research questions. The probing questions helped us make the conversations more detailed and concrete. We also asked other probing questions (not in the table) about achieving quality in a Scrum environment because the probing questions varied across participants and were influenced by the interview dynamics.

Our participants were distributed geographically, so all interviews were conducted using Zoom, an audio-video conferencing tool. The interviews lasted 40–90 min with a total of 41 hours and 22 min of audios. The audios generated a total of 683 pages verbatim after transcribing (an average of 17 pages per interview). The first author conducted the interviews in the period between January and August 2020. We used "Temi", an online transcription tool, to transcribe the interviews. However, this method does not always yield accurate transcription. Hence, we manually checked the transcripts and made the necessary corrections. Once an interview transcription became available, we sent the transcript to the participant for review. This step of the process was intended to validate the transcripts by our participants. This technique empowers the participants to check, "is that what I said?" (Mero-Jaffe 2011). Twenty-one participants advised no corrections, and four came up with clarifications for some of their statements. Some of the remaining participants either advised no need to check (eleven) or did not respond (seven). We anonymized[1] the interviews and made them available here.[2]

3.2 Phase 2 (RQ1 & RQ3)

In Phase 1, we collected Scrum practitioners' perspectives and experiences with a focus on positive outcomes. Subsequently, in Phase 2, we aimed to corroborate and validate the claims made by Phase 1 participants using negative cases, i.e., cases in which the outcome was not positive; "not all ravens are black", i.e., why some Scrum software development teams do not achieve better software quality? Considering positive and negative cases with different outcomes makes study design more rigorous and enhances the conclusions' validity (Emigh 1997; Hanson 2017). This method is more powerful than using only agreements because the divergent outcome permits causal conclusions (Emigh 1997). Variants of negative case methodology are commonly used in social research (Emigh 1997); however, this is not common in software engineering qualitative research. Furthermore, when negative cases are explained, the general explanation of successful cases is strengthened (Mahoney and Goertz 2004).

[1] All Phase 1 participants agree for their anonymized interviews to be made available part to the study report.
[2] https://doi.org/10.5281/zenodo.6624063

We opted for case studies for this phase. The purpose of a case study is to understand complex social phenomena, focusing on specific cases in depth, and to retain a holistic, real-world perspective (Yin 2018; Runeson and Höst 2009). The purpose of Phase 2 is explanatory, seeking an explanation of a particular phenomenon in its natural setting (Runeson and Höst 2009). We sought to understand why two Scrum teams did not achieve better software quality.

In Phase 1, the method followed was indirect. The researcher did not have any means to examine the accuracy of the data supplied by the participants. Participants chose how to answer the questions and shared their experiences related to the questions with the researcher. On the other hand, in observations, the observer can directly check the accuracy when observing. In studying negative cases, observations are more reliable than interviews or questionnaires, requiring less active cooperation and respondents' willingness (Mahoney and Goertz 2004). Often some respondents do not like to speak about their problems, struggles and failures to an outsider. This reluctance of respondents is much less likely to occur when the researcher spends some time together with the people being studied as they go about their daily activities, for the researcher may observe things that might not be reported in an interview (Mahoney and Goertz 2004). Thus, observational research can provide a richer understanding of software engineering practice, especially of its collaborative, social and human aspects (Sharp et al. 2016). The cases subject to this phase study are two software development projects carried out by two companies: an information technology and services provider and a technology startup. Both projects used Scrum for the projects to deliver software for external clients. Our units of analysis are the projects' teams and their processes. The contexts are the two companies.

3.2.1 Case Selection

Our cases are "typical." Yin (2018) explains that a "typical" case study exemplifies the investigated phenomenon and represents similar cases. The study's objective is to look within the case instead of comparing it with other cases (Yin 2018). Cases are often selected based on availability (Benbasat et al. 1987; Runeson and Höst 2009). We selected two companies as the subject of our Phase 2 study. In this study, we will refer to the two companies as **Kolkata** (located in India) and **Phoenix** (located in the USA) in this report. We opted to name them by the names of their respective cities to maintain their anonymity. For the selection of companies, we approached a "consulting firm" (i.e., the recruiter) specialized in assisting software development companies in implementing an "agile transformation" or improving an agile implementation that is not performing satisfactorily or according to expectations. They helped us recruit **Kolkata** and **Phoenix**. Relevant variables should be used in case selection (Runeson and Höst 2009). We informed our recruiter regarding our selection requirements, which are described in Table 6. The "+" sign in the second and third columns indicates that the selection requirement is met. In the last column, we share the evidence used to validate the selection criteria.

As a part of the recruitment process, we had two separate meetings with the CTO of **Phoenix** and a program manager from **Kolkata**. When asked about their respective Scrum implementation, the CTO replied, "could be better", and the program manager answered, "we are struggling ... We are learning and trying to find what works for us." When asked whether Scrum had helped achieve or improve software quality, the CTO said: "the same problems still exist. More and more bugs appear in UAT [User Acceptance Testing]", and the program manager replied: "we are not better than before [adopting Scrum] ... We produce so many bugs during the Sprints and our clients find more bugs in UAT [User Acceptance

Table 6 Selection requirements for the cases

Selection requirements	Kolkata	Phoenix	Evidence used to validate the selection requirements
The company uses Scrum for software development projects	+	+	The recruiter confirmed this requirement. During the initial recruitment meeting with the companies, they confirmed to us that they are using Scrum.
The company has been using Scrum for at least two years	+ (2 years)	+ (3 years)	Prior to the recruitment meetings with the companies, the recruiter confirmed this requirement. We validated this information during the recruitment meetings.
Scrum implementation has not helped to elevate achieving better software quality	+	+	The recruiter advised us that the companies' objectives were to improve the quality of their products and their teams' efficiency using Scrum. During the recruitment meetings, both companies indicated that their Scrum implementations did not help them to achieve these objectives.

Testing]." Their testimony raised our confidence in the suitability of both cases for the purpose of Phase 2.

The scope of our engagements with both cases was to gain access to the teams and become observant of their daily activities, ceremonies and collaboration channels (e.g., Skype (**Phoenix**) and Microsoft Teams (**Kolkata**)). Part of the agreement also allowed us access to the team's software development tools (i.e., JIRA (**Phoenix**) and Azure DevOps (**Kolkata**)). The first author carried out all observation activities. Both companies asked him to sign a non-disclosure agreement (NDA) that specified that no information or data collected during the study could be released publicly except anonymized quotes and screenshots.

3.2.2 Case Description

Kolkata **Kolkata** is a privately owned company incorporated on 8th January 2010 in **Kolkata**, India. Directors and key management of the company have authorized a capital of US$10M to finance the operations. The services performed by the company include software development advisory, software migration, and hosting to support and elevate the core competitiveness of businesses. The company offers software development services to various industries, including finance, logistics, marketing, business development, and human resources. The company has 225 employees with "sale" teams located in the US, UK, and Australia.

We were assigned to a project in progress. The project was about developing new software for an external client. We commenced the team's observations at the start of the 36th Sprint and continued for three months. The client was an American multinational corporation incorporated in the USA and founded in 1906. It produces starch, glucose syrup, and high fructose syrup. Its annual revenue is US$5.84 billion (in 2019), with a gross income of US$703 million and net income of US$454 million. The total assets of the company are estimated at US$5.73 billion with total equity of US$2.41 billion. It has 11,000 employees and operates in 44 different locations around the world.

This client commissioned **Kolkata** to develop software to manage its workflows used to produce customed ingredients solutions, e.g., sweeteners, starches, nutrition ingredients and biomaterials that manufacturers use in everyday products from foods, beverages to paper and pharmaceuticals.

Phoenix **Phoenix** is a company based in Phoenix, Arizona (USA). It is a technology startup. It has thirty-five employees, and it generates US$1.33M in sales, according to Dun and Bradstreet. The company was incorporated on 9th May 2008. It develops and maintains software for clients from diverse industries.

We were assigned to a new project that had not been launched then. We started the observations from the beginning of the project. The client was a digital media company operating from Austin (USA) with 50 employees. It began its operations in 2001. The company develops digital solutions for their clients across the US. The engagement with **Phoenix** consisted of developing a Mobile App to become a desired destination for wine lovers to discover and purchase wine.

3.2.3 Data Collection

Qualitative data are frequently used in case studies because it allows the collection of detailed and extensive descriptions (Runeson and Höst 2009). We used two data collection methods: direct observations and software development artifacts analysis (e.g., User Stories, Sprint Planning Board and defects reports).[3] We made the data analysis available

Direct Observation The first author conducted the observation activities for three months (November 2020 to January 2021). These observations were a "low degree of interaction by the researcher" (Runeson and Höst 2009). In this type of interaction, the researcher is perceived as an "observing participant" by the subjects, with the latter having "high awareness of being observed" (Runeson and Höst 2009). The researcher attended various project meetings and Scrum ceremonies. In addition, he was given access to the teams' discussion channels (i.e., Skype (**Phoenix**) and Microsoft Teams (**Kolkata**)). All meetings and Scrum ceremonies were audio-recorded, and immediately after the event, the researcher would listen to the recording and compile a field note for each event. This allowed the researcher to focus on what happened during the event and reflect on it later while writing the field notes. During this period, the USA and India were experiencing "lockdowns" due to the COVID-19 pandemic. Both teams were working from home and using virtual meeting tools to collaborate and communicate. Hence, the researcher was able to conduct the observations from Copenhagen (Denmark). In addition, both companies were in two different time zones, which facilitated the observations to be carried out in parallel. Table 7 summarizes the events observed and the number of occurrences, i.e., how many times the researcher observed a particular event.[4]

[3]The agreements entered by the first author with both Phase 2 companies prevent us from making Phase 2 data and analysis available.

[4]An agreement with both cases was reached to engage, record and use informal conversations with team members. **Kolkata** communicated this agreement to the team via email and **Phoenix** in an introduction meeting of the researcher to the team. At the start of each informal conversation, the researcher made the participant aware of the recording of the conversation and her right to object to participation and/or recording. The researcher also read a brief statement of how the conversation data shall be used in the research process.

Table 7 List of direct observations

Cases	Events	No. of observations	Examples from fieldwork
Kolkata	Daily Scrum Stand-up	48	Fig. 6 in Appendix 1
	Sprint Planning	9	–
	Sprint Retrospective	10	Fig. 7 in Appendix 1
	Requirements Clarification Sessions	21	–
	Teams' discussion channels (Ms Teams)	38	–
	Software Development Artifacts (Ms Azure tools)	24	–
	Informal conversations with team members	36	–
Phoenix	Team Huddle (similar to Scrum Stand-ups)	32	–
	Sprint Planning	11	–
	Sprint Retrospective	7	–
	Teams' discussion channels (Skype)	41	–
	Software Development Artifacts	18	Fig. 8 in Appendix 1
	Informal conversations with team members	24	–
	Internal Product Demos	11	Fig. 9 in Appendix 1

Software Development Artifacts The first author was given access to software development artifacts and tools used in both cases. **Kolkata** uses Microsoft Azure products, and **Phoenix** uses Atlassian[5] products. For **Kolkata**, the first author had access to these tools: Azure DevOps,[6] used by the team to track work progress, share code and release software; Azure Boards, used for tracking work plans, activities and discussing work among the team, and Azure Test Plan, used to plan testing and report defects. For **Phoenix**, the first author was given access to these tools: JIRA, used by the team to create user stories and issues, plan sprints, and collaborate around tasks; Confluence, used by the team to store and collaborate on knowledge artifacts (e.g., requirements document, "How to do" documents) and Zypher, used for software testing. Frequently, the researcher examined the content of these tools and compiled field notes accordingly. A systematic examination of these tools was not necessary, as the content was not updated frequently.[7]

3.3 Data Analysis

3.3.1 Phase 1 & Phase 2 Qualitative Data

We collected qualitative data during both phases: interviews and field notes. Therefore, our analysis method is the same for both data sets. We opted to use Miles et al. (2014) and Saldaña (2021) guidelines. We made this choice because the guidelines are comprehensive

[5] https://www.atlassian.com/

[6] https://azure.microsoft.com/en-us/services/devops/

[7] The researcher continued to have access to the projects artifacts (e.g., Jira and Azure Boards) which has been agreed with both companies, to remain updated with the project progress until the closure of the projects. The agreements with both cases also covered continuous access to team members for occasional informal conversations until the end of the projects. Even though, the observations ceased in January 2021, the first author had frequent contact with the companies through the point of contacts for projects' updates.

and enable deeper analysis. Miles et al. (2014) and Saldaña (2021) propose two stages of analysis: (1) *First Cycle* and (2) *Second Cycle*. The data analysis of both phases is available her.[8]

First Cycle During this coding phase, codes are assigned to data "chunks" (Miles et al. 2014). Coding is the "condensing" of data into meaningful labels relating to a particular research question (Miles et al. 2014). The purpose of this phase is to assign codes initially. Then, in the subsequent phase, they are used to identify reoccurring patterns. We used an inductive approach to our coding. This is a ground-up approach where we derived our codes from the data without preconceived notions of what the codes should be. We opted for "descriptive", "In Vivo", "process coding", "value coding", and "causation coding" methods (also referred to as code types) (Miles et al. 2014).

As Miles et al. (2014) explain: "a descriptive code assigns labels to data to summarize in a word or short phrase." On the other hand, in "In Vivo", codes use the participant's own words (Miles et al. 2014). The intent is to preserve the authentic meaning of what the participant has conveyed. "Process coding" conveys action in the data. The labeling of these codes typically are gerunds that end with "ing." Miles et al. (2014) explain that "value coding" identifies participants' values, attitudes, and beliefs. A value is the importance that a participant attaches to ideas and other things (Miles et al. 2014). An attitude is how a participant thinks and feels about ideas or things (Miles et al. 2014). A belief includes a participant's knowledge, experience, opinions, and interpretations of the social world around the participant. We also used "causation coding" to locate, extract, and infer causal beliefs from our qualitative data. Miles et al. state: "this method [causation coding] extracts attributes or causal beliefs from participant data about not just how but why the particular outcome came about. The analyst searches for combinations of antecedent and mediating variables that lead toward a certain pathway" (further details on how we used this method are explained in the Causal explanations section below). We opted for this combination of coding methods because our initial examination of the data (i.e., reading through the data to get a sense of what it looks like) indicated that our participants, in some instances, used words to directly refer to an existing construct in software engineering or Scrum (e.g. Fit for purpose, collaboration and accountability) while, in other instances, they discussed a concept by explaining it or providing examples. They also implied actions in their accounts of their experiences. Miles et al. (2014) suggest further coding methods, e.g., "emotion coding" and "magnitude coding." However, these are not relevant to the data we collected.

The first author conducted the first round of coding. Then, the second author reviewed the codes and provided feedback and suggestions for additional codes. Subsequently, the first author reviewed and decided on a final list of codes; this allowed us to probe the first coding round and consolidate our decisions on codes. Miles et al. (2014) explain that this is a "reliability check" where researchers must reconcile their differences for more credible conclusions.

Second Cycle Saldaña (2021) explains that this cycle synthesizes the various codes of the first cycle into a more comprehensive and unified scheme. This cycle is referred to as "pattern coding" (Miles et al. 2014; Saldaña 2021). It is a process of grouping codes into a smaller number of "meta-codes" (Miles et al. 2014; Saldaña 2021). First Cycle codes are "transformed" into pattern codes. These patterns are clustered either by code types or by

[8] https://doi.org/10.5281/zenodo.6624063

codes on the same topics, concepts, or logically mean the same thing (Miles et al. 2014). Table 8 shows **RQ1** Pattern Codes, some of their corresponding First Cycle codes, and examples from the data.

Causal Explanations Our research question **RQ2** aims at exploring whether the Scrum method and its principles and values (e.g., transparency and accountability) as promoted in the Scrum guide (Schwaber and Sutherland 2017), enable, in any shape or form, achieving software quality. This implies testing whether Scrum, as a process and system of values, exerts an influence on achieving software quality and in particular how and to what extent. Maxwell (2012) asserts that causal explanation is a "legitimate" and significant goal for qualitative research. Miles et al. (2014) explain that causation is identified during coding by looking for combinations of antecedents, mediating variables and outcomes, i.e., "casual chain". It is a three-part sequential process: ***antecedent → mediating variable → outcome*** (Miles et al. 2014). Seeking causal explanations is an iterative exercise (Miles et al. 2014). We mapped the initial causal chains in the First Cycle of coding, and then in the Second Cycle, we refined them. We did so because the three variables of the causal chains can either

Table 8 Example of **RQ1** pattern codes

Pattern codes	First cycle codes	Examples from the data
External	Conformity to business needs	*"Okay, quality to me. The biggest one is that it does what the user wants it to do. And I don't mean this behind the hood, I mean, just sort of like it produces the end result." (P32).*
	Free of defects	*"I think external quality is a product free of defects and It's the ability to fulfil and meet the requirements of the end user"* (P3).
	Usability	*"Then there's the sort of external facing quality, which is more, I suppose, a perceived perception of quality ... which could be from anything from UX experience through to performance. How clients actually interact with the system and perceive the system ..." (P37).*
	Add business value	*"The software should add value to the business"* P(19).
Internal	Sustainable design	*"Agile is keen also on internal quality because we believe in responding to change. So, we like to have good code quality and a sustainable design to cater for future changes" P(26).*
	Clean code	*"Quality on long term projects might also mean that you are able to handover your part of the project to someone else in a clean manner. So, the code is commented, the code is beautiful, I would say, this might mean for them quality code. Because if they keep bringing on new programmers, and there's no quality code, after a while the project will crash from start from zero"* P(36).
	Code maintainability	*"I think it's important to have something that is reasonably elegant from a maintainability and extensibility perspective. So, if we then the question boils down to if we consider maintainability and extensibility aspects that determine quality or improved quality, then yes, I would say that it's true"* P(33).

be a code (First Cycle output) or a Pattern code (Second Cycle output). We use this annotation ***antecedent → mediating variable → outcome*** in Section 4.2 of the findings to present the causal chains we identified in response to **RQ2**. For example, we found that collaboration (i.e., antecedent) promotes knowledge sharing (mediating variable) with an effect on software quality (outcome), ***Collaboration → Knowledge sharing → Software quality***. Every causality is presented and explained in detail in Section 4.2. Our analysis of **RQ3** expanded this causal analysis by exploring how contextual conditions hamper each of the causal links uncovered in Phase 1. To code these effects, we used tables (see, e.g., Table 10) that mapped, for each case, how the causal paths uncovered in phase 1 were affected by contextual conditions salient in the case.

3.4 Validating the Findings

Miles et al. (2014) use the term "testing or confirming the findings", while other authors prefer "validity." We opted to validate our findings using "feedback from participants" (Miles et al. 2014), also known as "member checking" (Creswell and Miller 2000). Lincoln and Guba (1985) describe this technique as "the most crucial technique for establishing credibility" of qualitative studies (Lincoln and Guba 1985). This process consists of presenting the interpretations back to the participants in the study to provide them with the opportunity to confirm the validity of the findings. We compiled a set of MS PowerPoint slides with a detailed description of the findings of Phase 1. Then, we organized two focus groups with developers and other participants from Phase 1 to discuss the findings. The details of this process are fleshed out in Section 5.

4 Findings

In this section we use N to make reference to the number of mentions in total in the data for a particular causal chain.

4.1 RQ1: Defining Software Quality from Scrum Practitioners' Perspective

Our informants extensively discussed two aspects of software quality: *"external"* and *"internal"*. While the external aspect relates to the impact on end-users, the internal aspect is defined by the qualities of the software design and the code and its ability to sustain future changes and adapt to business needs.

4.1.1 External Quality

Our participants used this categorization to refer to the correctness and the value of the software as perceived by the end-users. Two salient attributes were used to characterize this aspect of software quality: conformity to business needs (N = 35) and absence of defects (N = 32). Participant 33 provided a close to a textbook definition for conformity. He stated: *"but I think quality can best [be understood as] ... does the artifact or the product you developing achieve the required, the desired requirements, both functional and non-functional requirements expected by the target users?"* (P33). Some participants stressed adding value (e.g., P2 and P5) as an additional attribute to this category. Participant 2 has even made a clear demarcation between conformity and adding value, e.g., *"product quality can be simply defined as being defects free and add a business value or at least meets the user needs"*

(P2). This may imply the expectations of additional features or economic value that the software adds to the business, its processes and services.

Across the board, there is an almost universal agreement that the absence of defects is a fundamental attribute of external quality. Participant 31 used *"obvious"* to stress this attribute embodiment. He asserted: *"so, if you asked me what quality is, I mean, the obvious thing that people think about when they talk about software quality is probably bugs. So, there's something goes wrong, computer does something you didn't expect, and so on. That's the obvious, quality mission"* (P31). Participant 35 claimed an uttermost endorsement of this attribute, *"we have a zero-bug policy preferably we try to stick"* (P35).

4.1.2 Internal Quality

This category was used to refer to the internal workings of the software, which is often not visible to the end-users. Some of our participants used the term *"structural"* (e.g., P11, P20 and P21) to define internal quality. Participant 2 explained: *"internal quality has to do with the way that the software has been constructed or built. It has much more concrete attributes like clean code, simplicity, component reuse and flexible design."* Participant 32 used the analogy *"behind the hood"* to imply software qualities not visible to the end-users. There was a consensus amongst Phase 1 participants that internal quality should at least exhibit two traits: sustainable design and code quality.

Even though various terms were used to imply sustainable software design, e.g., *"scalability"*, *"extensibility"*, and *"flexible design"*, most participants did not come short of defining it. Accommodating changing business needs is the end goal of a sustainable software design, *"so, we like to have good code quality and a sustainable design to cater for future changes"* (P26).

Phase 1 participants used different terms and emphasize different dimensions of code quality. Some used *"clean code"* instead of code quality to imply the same thing, e.g., *"we focus on code quality or clean code"* (P29). Other participants made the effort to cite some of the desired features of code quality, such as *"readable"*, *"maintainable"*, and *"simple."* Still, there was no consensus on which of these features are more important to code quality. For example, while participant 32 emphasized readability of the code, participant 37 stressed maintainability. Participant 32 explained: *"but it is the biggest important one, if we're moving to looking like behind the hood, the biggest important thing with like quality code to me is making sure number one, that it's readable"* (P32). *"One is the quality that comes from not introducing any more bugs into a system, as well as continually improving your system without introducing any more issues. So that's sort of the quiet code quality SDLC maintenance quality aspects"* (P37).

While external quality was a prominent theme in the data from both Phase 1 and Phase 2, internal quality was more prominent in Phase 1 data, with Phase 2 observations showing little insight into perceptions of internal quality in the two cases (**Kolkata** and **Phoenix**). Still, we observed that **Kolkata's** team was keen on code review, which is a practice to assure code quality and identify defects early on the development process. In comparison, **Phoenix** relied only on documented code standards to promote writing quality code amongst its software developers. Overall, both cases focused on meeting the client's expectations on quality, which evidently accentuates the end-user perspectives. This focus on external quality was also influenced by the contractual agreement with the clients. In both cases, **Kolkata** and **Phoenix**, the requirements specification documents and the test cases covered some non-functional requirements such as "performance" and "security" but no mention, for example, of internal quality properties such as code quality, maintainability or scalable design.

4.1.3 Comparing our Findings to Industry Standards

It is insightful to compare these categories to established standards and definitions of quality. ISO/IEC 25010 defines software quality as "the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value" (ISO/IEC 2011). This definition focuses on the external quality aspect of software quality suggested by our participants. The ISO/IEC 25010 model also proposes additional characteristics relevant to external or internal quality not present in our data. The model suggests eight categories for product quality: "functional suitability", "performance efficiency", "compatibility", "usability", "reliability", "security", "maintainability", and "portability" (ISO/IEC 2011). Without diving into the discussion and the definition of these categories, as they are self-explanatory for a software engineering audience, we would like to point out that some of the model categories, i.e., "compatibility", "reliability", and "portability", were not discussed by Phase 1 participants nor were they observed in Phase 2 cases.

Against the emphasis on external quality in the ISO/IEC 25010 model, it is remarkable how strongly our participants in Phase 1 emphasized internal quality, especially sustainable software design and, thus, code that supports the continuity of the software. Participant 30 explains: *"when you give one developer the work that was done by another developer, it [should] scale[s]. It takes so much time for that new developer to understand what's being done. And something that is supposed to take us through our weekly sprint, takes about two weeks just because a new developer is trying to understand what was before. So, for us, code quality also means scalability. If we grow this team, or if we grow the product, it doesn't start crumbling."* Although sustainable software design may be related to the maintainability dimension of the ISO/IEC 25010 model, the interview statement highlights that facets of sustainable software design, such as easily understandable code, matter not only during maintenance but also during development. This may be partly due to the high amount of change and flexibility needed during development, as valued in "Principle 2" of the agile manifesto (Beck et al. 2001), which states: "welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage." This explains the affinity to accommodate future business needs by having a sustainable software design. The strong emphasis on sustainable design may also be inspired by "Principle 9", which states: "continuous attention to technical excellence and good design enhances agility." Even though "good design" is vague, practitioners seem to interpret this recommendation in the right direction to accommodate the requirement of "Principle 2."

The purpose of **RQ1** was not to "reinvent the wheel" but to establish a reliable and relatable definition as a foundation for examining **RQ2 & RQ3**. In addition, we wanted to explore how definitions evolved in light of changes in the ways software is developed when teams use Scrum. Our key finding in this regard is the strengthened focus among Scrum practitioners on internal quality, especially on sustainable software design, although we also observed that the projects examined in Phase 2 did not share this focus. It is interesting to note that, despite the abundant literature on the topic and numerous industry standards and definitions, some of our participants (e.g., P1, P3 and P5) maintained that software quality remains a subjective end result. Some even claimed, *"I know it when I see it"* (P3) and *"but to be honest with you in reality, we don't know it until we see it and agree this is the quality we want"* (P22). Still, our Phase 1 participants did not shy from defining it; they asserted that the core attributes are conformity to business needs and free of defects. This has become apparent in Phase 2 data; both **Kolkata's** and **Phoenix's** teams aimed at meeting their client's expectations for quality, defects reported during the Sprints were mainly misalignments with the stated business needs.

4.2 RQ2: The Value-Add of Scrum for Achieving Software Quality

Recall, **RQ2** seeks to understand how Scrum values, principles and prescribed events and activities enable teams to achieve software quality in any shape or form. Our analysis suggests that Scrum enables this in two major ways. First, Scrum values and principles support four social antecedents that strengthen the team's ability to produce quality software. These antecedents are collaboration, psychological safety, accountability, and transparency. Second, the events and activities prescribed by Scrum yield two process-induced advantages that help achieve software quality: iterative development and formal inspection and adaption.

All the social antecedents and process-induced advantages have emerged inductively in our analysis. That is, we did not use a pre-defined theory or known constructs relevant to Scrum or agile in order to find these antecedents and how they influence achieving software quality. We opted for an emergent strategy instead of testing known constructs in the literature to allow findings to emerge from the frequent, dominant and significant themes inherent in the raw data, without the restraints imposed by existing theories. This strategy allows flexibility and the freedom to create new knowledge (Miles et al. 2014). Even though most of the antecedents are discussed in the agile literature, the inductive approach facilitated the discovery of their effects on software quality, i.e., causal chains, which, to the best of our knowledge, have not been reported previously.

4.2.1 Social Antecedents of Software Quality

A social antecedent refers to a quality exhibited at the individual, team, or organizational levels. It also entails any process, practice or ceremony that is not technical (e.g., tools) or pertaining to software engineering practices (e.g., continuous integration) or known quality assurance practices (e.g., code review). Contrary to quality assurance and engineering practices, these social antecedents are complementary. Rather, for example, than intercepting defects or identifying possible violations of coding standards, they enable achieving software quality socially. This social enabling entails that each antecedent advances achieving quality by promoting a particular behavior, which has a consequent effect on achieving software quality (i.e., **Social antecedent → Behavior → Software quality**). For example, in the case of psychological safety, a social antecedent of software quality, the behavior is that team members care about quality by speaking out about quality and making efforts to increase quality without fear or feeling of guilt. The resulting effect is that errors and defects are pointed out and that the efforts necessary for achieving quality are invested.

Figure 2 depicts this synergy. Each social antecedent advances the ability of Scrum teams to achieve quality by promoting a behavior. The combined effect further advances the ability of the team to achieve software quality. Their absence, however, does not imply a categorical absence of quality, but rather the absence of a social trait that could further the team's ability to achieve quality. These social antecedents do not undermine or replace traditional software engineering and quality assurance practices (e.g., unit testing, static analyzers and continuous integration) for assuring quality but rather complement them. This Scrum value-add elevates the team's ability to achieve quality socially.

Figure 2 shows four causal chains, i.e., $v_1 ... v_4$. Each causal chain depicts the effect of one social antecedent and how it advances achieving software quality in Scrum teams. For example, transparency in Scrum teams promotes voluntary inspections of deliverables. These inspections produce an effect, i.e., feedback, which is invested in correcting and improving the deliverable. This causal chain is labeled v_4 in the diagram. We will use these labels

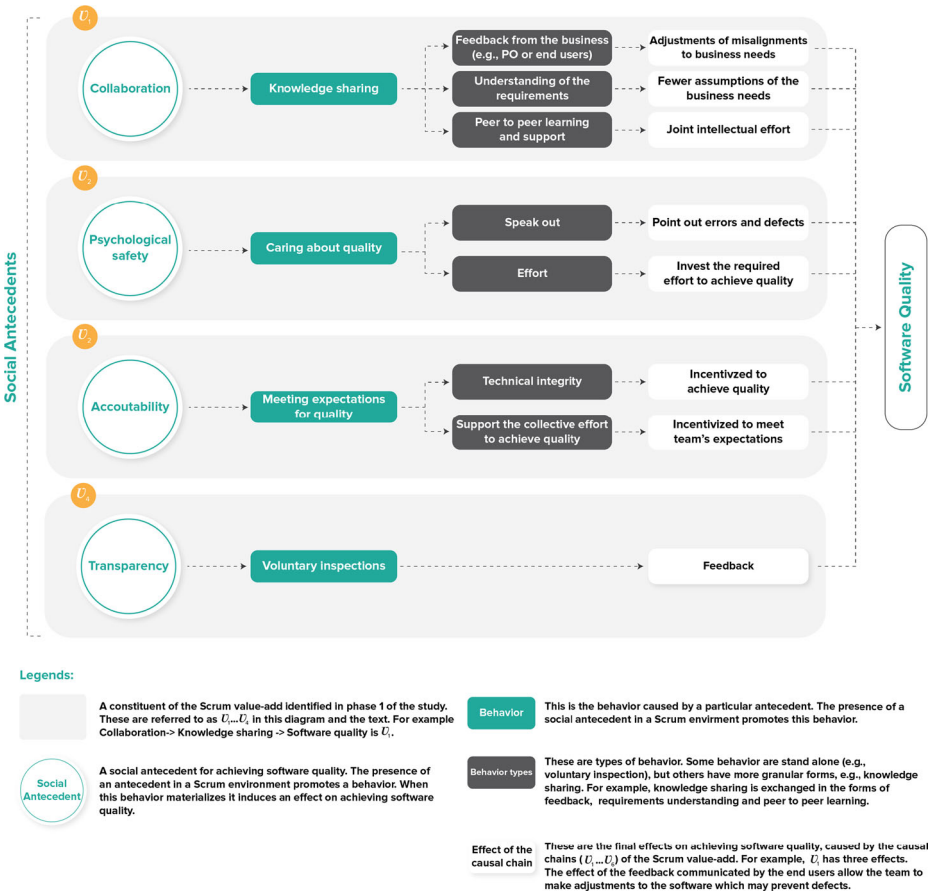

Fig. 2 Scrum value-add for advancing software quality—social antecedents of software quality

(v_1...v_4) in the upcoming text to make reference to the causal chains. We will also use them in reporting **RQ3** results to sync **RQ2 & RQ3** and demonstrate how these causal chains failed to materialize in both case studies.

Table 9 lists the causal chains we identified in our data. While the last column shows the total number of occurrences of each causal chain in the data (N), columns Mgt, PO, SM, QA and SD represent the spread of N across the various perspectives in our sample. For brevity, we used Mgt for management, PO for product owners, SM for Scrum Masters, QA for quality assurance and SD for software developers. Moreover, we used SWQ instead of software quality and inspection & adaptation without formal. The numbers evidence that our participants placed the greatest emphasis on collaboration and formal inspection and adaptation. The numbers also show that some antecedents are more relevant to some perspectives than to others. For example, software developers focused strongly on the effect of accountability on quality while product owners, scrum masters, and QAs did not discuss the theme. Similarly, it was almost only QAs and developers who highlighted the effects of process-induced advantages on quality. These numbers convey a message which is that quality assurance

Table 9 The causal chains and the number of their occurrences in the data

#	Causal chains	Number of occurrences in the data					
		Mgt	PO	SM	QA	SD	Total
Social Antecedents							
v_1	**Collaboration → Knowledgesharing → SWQ**	5	2	4	6	13	30
v_2	**Psychologicalsafety → Caringaboutquality → SWQ**	1	0	2	1	1	5
v_3	**Accountability → Meetingexpectationsforquality → SWQ**	2	0	0	0	7	9
v_4	**Transparency → Voluntaryinspections → SWQ**	2	2	0	0	3	7
Process-induced advantages							
v_5	**Iterativedevelopment → Modularization → SWQ**	0	0	0	0	6	6
v_6	**Inspection&adaptation → Continuousfeedback → SWQ**	0	0	1	7	8	16

professionals and software developers experience and appreciate the benefits of a development environment where these antecedents thrive.

We organized this section by social antecedents and their respective causal chains. We start a causal chain sub-section with a heading with a causal notation, i.e., **Social antecedent → Behavior → Software quality**. Then, we present each causal chain effect in Scrum teams and support it with data. At the end of each causal chain sub-section, we present a summary in gray boxes to concisely summarize the discussion.

Collaboration ⟶ Knowledge sharing ⟶ Software quality

Collaboration is an instinctive and well-established practice in Scrum teams. It takes place through various prescribed ceremonies (e.g., stand-ups and retrospectives). The ceremonies are not the only medium to collaborate. Our participants explained that Scrum helped them work "well together." For example, participant 12 attributed his team's success partly to collaboration, amongst other qualities. He stated: *"we were successful primarily because we worked together extremely well"* (P12). A similar claim was made by participant 28, *"Scrum helps us to work very well together"* (P28). Collaboration is advocated by the Scrum guide (Schwaber and Sutherland 2020, p. 11 & p. 14). It is further promoted by the prescribed "events", e.g., "Daily Scrum" and "Sprint Review." Our data show that in practice, software development teams interpret the intent of Scrum events as a requirement for collaboration to achieve a better outcome. This is apparent in participant 11's assertion: *"Scrum cannot be successful without team collaboration and teamwork. Even if one person wants to work in isolation, it becomes pretty difficult for that person. Scrum is a very inclusive process"* (P11). In line with this statement, many statements from Phase 1 participants documented that team interactions and knowledge sharing increased with the use of Scrum.

To decode this phenomenon, we looked in the data for a more granular explanation of collaboration. When our participants implied collaboration, they described a process where team members were expected to participate actively and engage equally. Scrum ceremonies encourage individuals to come out of their *"bubbles"* (P37) and silos, thus promoting access to and exchange of knowledge to ensure high quality of the software. Participant 37 explained: *"it's great to have developers sitting there and inside their own little bubble,*

developing code. If you're not able to talk to peers and get ideas on how to structure your code, and how best to solve problems, one, you can end up developing the wrong thing. You can end up writing code that, while it might achieve a goal, ... might be totally unmaintainable going forward. You might end up writing very inefficient procedures or stored procedures that are just ridiculous. You might end up spending days on a problem that your teammates could help you solve in one hour. ... For example, in talking with the team, especially the testers, they might help you pinpoint places in code that you haven't paid attention to ... Even when you're talking about your developments in your stand-ups, the testing team can also think of other ideas, the things they need to be testing. So, they're not in a bubble, either. I mean, there's all sorts of benefits to collaboration ... Or you definitely become better at writing code. You also become better at helping others write code, and helping, sort of helping the team work better together" (P37). Equal and early engagement of all roles in the team also helps break the *"bubbles"* (P37) and create fluidity in exchanging information, *"so being able to sort of open that that fluidity between the different pods, helps the whole work better"* (P37). Participant 19 (a QA engineer) explained the outcome of early engagement of the QAs in the process: *"we become better knowledgeable about the requirements, and we participate in the user stories. This collaboration helps us to know what to test and we catch more bugs"* (P19). Participant 38 summarized the synergy of this phenomenon: *"I mean, that in Agile, you have one team, and all knowledge is circulated within this team, and there is no need to pass this information. It's inside, it might be no formal documents, it might not be some website on Confluence [Knowledge sharing tool], or wiki. And you store that information inside as a team, and there is not any information loss when you move further"* (P38).

The rounded rectangle v_1 of Fig. 2 pieces together this social phenomenon. Scrum encourages cross-functional participation and equal engagement. For example, the Scrum guide describes the team as "a cohesive unit of professionals focused on one objective at a time, the Product Goal" (Schwaber and Sutherland 2020, p. 5). Our data show that Scrum organically promotes this "cohesive unit" by breaking barriers between functions, e.g., $developers \rightarrow QAs$ and $business\ users \rightarrow QAs$. Scrum ceremonies help teams break barriers by transforming them into a unified and cohesive unit. Participation becomes second nature for all team members when it comes to interaction, for example, in the daily Stand-Ups. Equal engagement in Sprint planning (i.e., every team member participates) creates inclusiveness. This social climate results in fluid knowledge sharing and has three distinct dimensions affecting software quality: feedback from the business, understanding of the requirements and peer-to-peer learning and support.

Feedback from the business users. Feedback from business users can be provided either through formal channels, e.g., during the Sprint reviews and the "client demos" (e.g., P11 & P19), or through informal interaction with users. The accessibility of the end-users, either via the product owner (PO) proxy or directly participating in the Scrum team, allows the team to have access to early feedback. The team capitalizes on this feedback to rectify errors and align the developed features with business expectations. Participant 31 explained: *"so, it's my experience is that the feedback impact on quality and on features is greater when you have access to the actual customer rather than someone who is simply the actual user, compared to someone who is the just the owner of the product. I think that's, that's very helpful if you can get access to the actual user and get quick feedback, loops, quick feedback cycles. So that definitely helps the product quality, because they also, always find something that you're rarely the main expert in whatever the application*

is supposed to be doing. If you're working on the same thing over many years, you may become one and get the sense of what the user needs. But typically, you need to have an actual user who has domain knowledge to point out the things you missed, which is also easier to do the sooner you get the feedback. This feedback helps to improve quality, errors are identified sooner then corrected" (P31).

Understanding of the requirements. The interaction with the business representatives permits continuous learning about the business needs, either through formal ceremonies, e.g., Sprint planning and Backlog grooming, or informally in daily encounters. Developers and quality assurance specialists (QAs) alike are less inclined to make assumptions about the requirements. Because of the participation of the business in the process and the fluidity of knowledge exchange, they are willing to validate their understanding of the requirements instead of making assumptions that could lead to defects. Hence, the effect of this behavior is preventive, i.e., it prevents defects by reducing the gap in understanding the requirements. Participants 4 & 6 assertions cement this interpretation, *"they [QAs] become intimately knowledgeable of the business requirements and the expectations. This helps testing the real users' expectations rather than making assumptions. This knowledge helps identifying and reducing bugs"* (P4), and *"I think the most important thing about Scrum is it brings people to collaborate better. I used the example of Sprint planning. During this meeting, the client, the developers and the QAs discuss and align their understanding of the requirements and expectations on quality. This significantly reduces bugs; developers know what to develop. They do not make assumptions and the QAs know what to test. Here you have it; a product with less bugs, that's quality"* (P6).

Peer-to-peer learning and support. Once barriers within the same function and cross-functions are reduced or even broken, individuals become more willing to help and less hesitant to ask for help. This fluidity creates peer-to-peer knowledge flow, which would remain largely untapped if people worked in silos. Then, a joint intellectual effort is invested in helping each other. This contributes to better coding and design decisions and helps tackle complex problems. Participant 37 explained this synergy passionately: *"If you didn't have that sort of knowledge sharing, they'd be sitting in a vacuum trying to figure things out, getting things wrong over and over and over again. And clients are just screaming. In the meantime, where we had some absolutely idiotic bugs just because people wouldn't talk to each other"* (P37). Participant 28 worked for a pharmaceutical manufacturing provider. He explained that his team develops *"complex"* software. Collaboration helped his team to overcome the complexity and had far-reaching effects; they collectively came up with enhanced coding decisions and design. He stated: *"Scrum helps us to work very well together. The product we developing is complex, so knowledge and how to deal with complex problem is important to us. We always discuss how to go about things and we come up with the most efficient solutions. You know, we develop automation software for pharmaceutical manufacturing; it can be complex. When we talk and learn from each other's we come up with better design and better code"* (P28).

To recapitulate, this social antecedent enables achieving software quality by making knowledge more accessible and shared voluntarily within the team. Then, the knowledge is invested in developing features aligned with the users' expectations and tested accurately. Knowledge sharing benefits the software's external quality (see **RQ1**) and contributes to the betterment of the internal quality when peer-to-peer learning occurs. Our participants reported that better decisions were made when the team invested their collective intellectual effort.

v_1

Collaboration ⟶ Knowledge sharing ⟶ Software quality

Collaboration is intrinsic to the Scrum method; it facilitates better knowledge sharing. The collaborative climate in Scrum teams accentuates the exchange of knowledge fluidly and frequently. This state helps advancing quality in several ways. The collaboration with the end users provides the team members with access to frequent feedback and better understanding of the requirements. While the former helps the team to take corrective measures to the features being developed to align the software with the business needs, the latter, reduce the tendency of the team to make assumptions about either missing information or their interpretations of the requirements. As claimed by our participants, this eventually reduces defects as assumptions can be wrong. The other aspect of collaboration is peer to peer exchange of knowledge. Our participants reported that the joined intellectual effort resulting from these interactions helps quality by collectively assisting each other's in figuring out the best decision for coding, design and resolving complex design and coding tasks.

PsychologicalSafety ⟶ Caringaboutquality ⟶ Softwarequality

Psychological safety is a shared belief that individuals feel safe about the interpersonal risks that arise concerning their behaviors in a team context (Edmondson 2018). Our participants used words like *"less afraid of failure"* (P3) and *"they feel it is safe to do so"* (P5), to imply the presence of this social antecedent in their teams. They reported that psychological safety influences the behavior of developers. In a psychologically safe climate, two effects emerge and advance achieving software quality: developers invest further effort on quality and speak out when quality is not within the expectations set by the team. These two effects are inherent to the behavior of caring about quality.

Our data suggests that developers care about quality and invest further effort to ensure it when they feel safe to do so. Project constraints such as schedule may exercise pressure to meet deadlines and compromise quality. However, when developers feel safe, they invest effort and time to ensure their deliverables' quality. This claim is evident in participant 5's assertion: *"developers invest on quality when they feel it is safe to do so. For example, if you blame them for not meeting the deadlines and you don't give them enough time to work on the quality to start with, then they will compromise quality to meet the schedule and avoid the blame. I've seen it happening; developers become disengaged and uninterested; then they produce crappy code"* (P5). This effect is not only observed within developers. Similar behavior is also observed amongst the QAs, according to participant 12 account: *"it does [psychological safety helps achieving quality]! What I have observed is developers and QAs for example take initiative more and invest on the success of the product including its quality. For example, developers put more effort and time on writing better code. The QAs become invested and care about the quality of the software, even the relationship between developers and QAs improves. They communicate better and share information faster to understand bugs. The team becomes more efficient at assuring quality. I saw improvement on the quality"* (P12).

Our participant also pointed to another benefit that psychological safety brings for software quality: Team members speak out when they think the quality of a deliverable is not within their expectations or the team's expectations and they feel psychologically safe. Participant 21 explained: *"but we are encouraged to speak our mind without fear. This helps a lot. For example, when I see work with lower quality than I expect or the whole team expect, then I'm not afraid to say it. If you were with a team, like if you work in a team that's engaged and people like care about what they're doing, then there's that psychological safety where*

sometimes you can make mistakes or other people make mistakes and know that that's okay. That's definitely a better feeling. Not only mentally, but I've seen people attitudes changes. They care about the quality of the product and this helps for better quality" (P21). Pointing out quality issues in a peers' deliverable may create a feeling of guilt. However, it is not the case in a psychologically safe environment because the relationships are *"stronger."* This was discussed by participant 29: *"bugs are found and communicated without fear or feeling of guilt. The team operate better, because our relationships are strong. We feel very close to each other's and minimum power over us. We invest more effort on quality and we produce code with less bugs"* (P29).

Promoting psychological safety in the development environment brings social qualities to the team, thus advancing its ability to achieve software quality. Developers invest effort in the quality of their work, and team members feel no fear to speak out when quality is not according to the expectations. However, this extent on quality is not unique to psychological safety. A simple gesture like involving the developers in the discussions with the client has a similar effect. Participant 30 explained: *"meeting [with the client] helps them [developers and QAs] feel part of the project and they're more willing to give that extra effort. It makes them more keen on ensuring quality because they don't feel like it's does some work I have to do. They're more, they feel let me do a good job"* (P30). He later reaffirmed the effect of not fearing the consequences on his productivity and producing *"better quality"*. He said: *"when things don't work as well, I don't feel like oh, no, it was my fault. Everything went bad because I made a mistake. It's like, because we're all in this environment of review. That review doesn't mean why did you get this wrong? It means I can see you didn't think about this. Next time let's do this. I don't feel attacked for being wrong. If that makes any sense. I feel less inclined to feel attacked. So, I'm more productive, because I can just focus on doing what I can to be more productive and better quality"* (P30).

The rounded rectangle v_2 of Fig. 2 summarizes this effect. The absence of fear from the eventual consequences of one's actions forges a behavior of caring about quality. This care implies investing the necessary effort to meet quality expectations. For example, developers spend sufficient time looking after the quality of their code because they are not afraid to do so. To speak out when the quality is not up to the expected standards is the result of being concerned which is an attribute of caring. For example, QAs do not feel the "guilt" of raising defects. Neither do developers because the risk of offending or implying blame diminishes when this sense of safety prevails. While investing effort in looking after quality has a preventive impact the software quality, i.e., prevent defects or unmaintainable code, on the other hand, speaking out has a corrective impact. The errors become known, and adjustments are carried out accordingly. Participant 32 explained it more agreeably: *"if you're living in fear of like the manager cracking the whip on you, you're living in fear of the manager cracking the whip on you. And that's not going to encourage you to do your best that's going to encourage you to cut corners including quality"* (P32).

Although some of our participants' accounts (e.g., P3, P13 & P21) suggest that psychological safety cannot be attributed to the use of Scrum, it can be argued otherwise. There is a parallel between *"to be less afraid of failure"* (P3) and "courage", which is advocated in the Scrum guide (Schwaber and Sutherland 2020, p. 4). It states: "Scrum Team members have the courage to do the right thing" (Schwaber and Sutherland 2020, p. 4). Moreover, the Scrum values of "respect" and "openness" (Schwaber and Sutherland 2020, p. 4) call for a team environment in which disagreements are dealt with in a constructive way, implying that team members need not be afraid of blame or offenses as a result of disagreements. Although endorsement of Scrum values may thus promote psychological safety, Participant 32's account indicates that these qualities are also the outcome of individual and team-level

learning processes: *"but I find that there is a, there's a taboo, when it comes to retrospectives around talking about the bad. And you can find yourself under fire quite a lot for being the person that has the courage to talk about the bad. Because unfortunately, people don't like to hear bad things. But one of the key things that I've learned in life is that talking about the bad things helps you grow and improve. That's the whole point of agile, you can probably see that I apply agile to like my own life at this point. Like I'm a bit of a nutter when it comes to that"* (P32).

v_2

Psychological Safety ⟶ Caring about quality ⟶ Software quality

Our participants reported that psychological safety in Scrum environments empowers team members to speak out and motivate them to invest efforts in meeting quality expectations. Speaking out entails providing voluntary feedback on the quality of the software artifacts and pointing out errors without experiencing tension by both parties. According to our participants, both the receiver and the provider of the feedback do not experience the feeling of guilt and are at ease to talk about shortcomings in quality. This advances quality because errors are pointed out and talked about. In addition the feeling of safety propels developers to invest the necessary effort on assuring the quality of their work. According to our participants, developers are motivated to produce quality deliverable, when there are no repercussions to do so.

Accountability ⟶ Meeting expectations for quality ⟶ Software quality

Accountability, as it emerged in our data, implies assuming responsibility for one's work and its quality. It also implies taking ownership and shared obligations, e.g., *"you are accountable to the rest of the team. See, the culture is that everyone is accountable for everyone. The rest of the team is accountable for you. And you hold accountability for other people too"* (P2) and *"they [team members] take ownership of the work they do and its quality"* (P5). As per participant 2 and 5 quotes, this social antecedent has two dimensions; while the former refers to collective accountability, the latter points to individual accountability.

Some of our participants implied that accountability is the product of being a self-governing team, e.g., *"I think it's [self-governing team] very powerful, but you're suddenly more, you're not responsible, you're accountable, you're more accountable to your colleagues, rather than to your manager. Because that's the whole technical and reputational thing in it as well"* (P31) and *"what Agile does is takes that responsibility and hands it to the team. Now, because the developers, the testers, the business analysts feel valued, because they know they an instrumental part of delivering this and this, they're not just worker bees. They take a personal responsibility and therefore provide a personal commitment in ensuring that whatever they do is improved quality. At least I'm speaking for myself, but I've seen this overall, within projects over the years"* (P33) or simply a commitment to a value advocated by Scrum, e.g., *"when the culture is that everyone is accountable for everyone, developers feel accountable to meet the team's expectations on quality"* (P2). Either way, this Scrum value has a positive effect on achieving software quality. It brings about the behavior of wanting to meet the team's expectations on quality. Our participants described this effect as not *"letting the team down"* (P2), *"reflection on them"* (P33) and *"you don't want to make team looking bad"* (P38) when the expectations are not met. This antecedent

may have more influence on software developers than other roles in the Scrum team. It appeared more prominently in the interviews with developers (N (developers) = 7 out of N (all sample) = 9) than in interviews with other roles (e.g., P2 and P5).

The rounded rectangle v_3 of Fig. 2 depicts how accountability influences advancing software quality based on our data. The behavior resulting from this antecedent has two strands: maintaining technical integrity and supporting the collective effort of the team to achieve quality. While the desire to maintain technical integrity stems from the individual feeling accountable, supporting a collective effort is related to collective accountability. Maintaining technical integrity, for example, was implied by developers using references to *"personal pride"* (P36) and *"reputation"* (P31). This desire to maintain technical integrity incentivizes developers *"writing good quality code"* (P38) and aiming for *"better quality"* (P33). Supporting the team in its pursuit for quality is the other strand of this behavior, e.g., not *"letting the team down"* (P2). In order to contribute to the team's endeavor to achieve quality, developers are incentivized to meet the expectations. Participants 36 and 31 endorse these conclusions, *"yes, it does [Scrum motivate me to write better code]. Personal pride for the retrospective and you present your results. And we are actually quite proud of your code, checks all the boxes. It's beautiful. It runs, it provides results. Because it was a simple task, you have a high chance of success to check all these boxes. You estimated enough time to perform all the required steps for the formatting, the commenting, the testing, the rechecking, and you actually are proud of your code. And that brings value your personal value, increases your morale, which helps you actually evolve and do better and better code all the time"* (P36), and *"if I'm in a team of peers, and I deliver something that has way too many defects compared to the others, it reflects poorly on me in the eyes of my peers"* (P31).

Scrum explicitly advocates for accountability. The Scrum guide states that "the entire Scrum Team is accountable for creating a valuable, useful Increment every Sprint" (Schwaber and Sutherland 2020, p. 5). Although accountability for the quality of the "increment" is not definite in the guide, it is somehow implied. The guide further states, "holding each other [developers] accountable as professional" (Schwaber and Sutherland 2020, p. 5). Developers in Phase 1 of our study explained that they account for the quality of their work because they believe it showcases their professionalism. This was explained by participant 31: *"technical and reputational thing"* (P31). The technical reputation is compromised, e.g., *"a reflection on them"*, if she does not meet the team's expectations on quality.

v_3

Accountability ⟶ Meeting expectations for quality ⟶ Software quality

Our participants (mostly the developers) reported feeling accountable for their work, in Scrum environments. Some explained that this quality is inherent to some level of self-governing they experienced or simply because of the empowerment promoted by equal participation and engagement. This quality implies taking ownership and responsibility of one's work. Developers, in our sample, explained that they become incentivized to maintain their technical integrity and support the collective pursuit for achieving quality. Developers equate the quality of their code to their professionalism, integrity and showcasing their competences to gain peer recognition. The other implication of feeling accountable is the desire to not "let the team down." Meeting the expectations of the team on quality becomes a driver for developers to contribute to the collective endeavour to achieve quality.

Transparency ⟶ Voluntary inspections ⟶ Software quality

As we understand it from our data, transparency is the equal access and visibility of work information and progress. According to our participants, Scrum teams aim to communicate openly, and honestly to make information flow freely within the team. Participant 37 described his team environment as "open" and "frank", *"it's very open. It's very frank"* (P37). Transparency is one of Scrum's "pillars", in addition to inspection and adaptation (Schwaber and Sutherland 2020, p. 3). According to the Scrum Guide, transparency is intended to allow inspections of the team's deliverables. The guide states: "transparency enables inspection. Inspection without transparency is misleading and wasteful" (Schwaber and Sutherland 2020, p. 4). Our data showed that this intent is materialized in practice, as reported by our participants. Transparency advances software quality in Scrum teams by allowing team members to inspect the quality of each other's deliverables.

Scrum ceremonies facilitate this social antecedent. For example, during stand-ups, team members become aware of each other's tasks. When an artifact is available for review and retrospectives, it allows for in-depth inspection of Sprint's achievements. Our participants implied that transparency is an inherent quality of Scrum. Once the Scrum method is operationalized and adhered to, transparency becomes intrinsic to the team's work environment. Participants 1 and 31 described this phenomenon as not being able to *"hide"*, implying that team members' deliverables, including their quality, are visible. Participant 1 stated: *"in Scrum, you don't have the luxury of hiding for six months"* (P1). Participant 31 added: *"so, it gets harder to sort of hide behind and make excuses for bad performance as well, in this case. It quickly becomes more apparent if people in the team aren't pulling their weight, say or if they're doing subpar work"* (P31).

This antecedent encourages voluntary and informal inspection of the quality of deliverables, which advances the quality of the software. Participant 3 explained the effect of transparency on his team after they adopted Scrum: *"I think the Scrum environment helps in that way because everyone was more transparent. Everyone was less protective over their code. We also had one product backlog, one repository, everyone could see anyone's code or change anyone's code. That also helped with quality. Let's say I code something, and my code has mistakes or whatever, anyone was empowered to comment on my code and help on my code and submit approve request to better it. Of course, the person who wrote it would have to approve it but generally it would be approved"* (P3). Participant 28 affirms this claim: *"people can comment on my work anytime and criticize my coding and design decisions. This helps the quality of my work and I learned extensively from my teammates feedback. As I said I'm the youngest and less experienced in the team"* (P28). The effect of this social antecedent on software quality is corrective. Whether deviations from code standards or specifications, errors are identified by peers and then pointed out for adjustments.

v_4

Transparency ⟶ Voluntary inspections ⟶ Software quality

Transparency is materialized in Scrum teams when all team members have equal access and visibility to software artifacts, deliverable, decisions and the daily operations of the process (e.g., user stories progress, code released for review, architectural decisions and changes, etc.). Our participants experienced that this quality instigates team members to voluntarily comments, provide feedback and suggestion to improve quality, whether it is the code quality, a design decision or how to resolve a particular defect.

4.2.2 Process Induced Advantages

The essence of Scrum is an incremental approach to software development. It advocates creating value through iterative increments to accommodate unpredictability and mitigate potential risks. Scrum also recommends frequent inspections to identify and address deviations from the customers' expectations, i.e., adaptation. Our data show that these process-specific propositions induce advantages that stimulate achieving software quality. While the social antecedents arise from a team espousing Scrum values and principles, iterative development and formal inspection and adaptation arise from teams following the activities and events recommended by Scrum. Once operationalized, they become ingrained practices. Figure 3 illustrates how the process antecedents promote advancing software quality. Similarly to Fig. 2, we used v_5 and v_6 to label the causal chains of iterative development and formal inspection and adaptation. In the subsequent text, we shall use these labels for references.

Iterative development ⟶ Modularization ⟶ Software quality

Iterative Development Scrum recommends that the product should be developed in iterative Sprints. Each Sprint has a specific goal with the purpose to "increase the product value" (Schwaber and Sutherland 2020, p. 8). Our participants, especially software developers, praised the effect of this style of development. They explained that it brings modularity to the tasks, which gives the developers more control and stronger focus over a contained part of the code and its quality. As a result, developers become more confident and willing to consider and implement actions that elevate the quality of the code. Participant 32 explained: *"it's like it [iterative development] creates this implicit relationship between good readable code and quality code where it's sort of like, you know, saying, like, create this one bit that does this one thing, it is separate from everything else. Because, you know, this was that was the term I'm looking for. It's back to that sort of like the temptation to just lump things all together in the same index file or whatever. And then you end up with like, very messy*

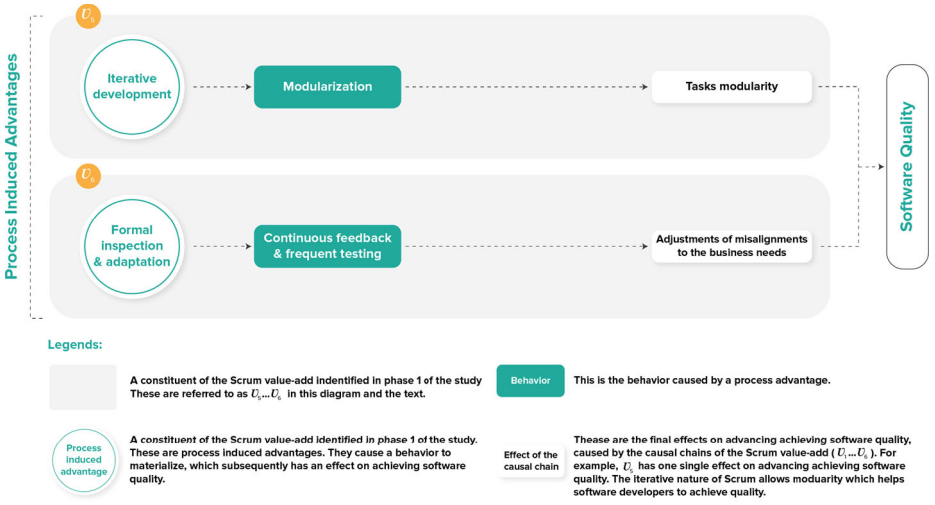

Fig. 3 Scrum value-add for advancing software quality—process induced advantages

and chaotic code. Whereas in this approach, you can literally just map it out really neatly" (P32). Modularity enables devotion and focus on the task or a specific portion of the code, and careful attention is given to its quality. Participant 36 used *"focus"* and participant 38 *"concentrate"* to explain this effect, *"I think Scrum helps in this regard the code quality because it helps you focus on your specific code"* (P36) and *"so, when you have well defined sprints, you work on this task only, and that helps you to concentrate and try good quality software"* (P38). Then, the behavior resulting from having this process feature is the ability of the developers to focus on the quality of their code because of the atomicity of the coding tasks. Participant 30 put it in a forthright manner: *"well, the less code I have to write, the more I can focus on doing it better. That's my short answer. The less code I write, I focus on how I might make my code better. I always check its quality. The less code, the more I can focus on quality"* (P30).

v_5

Iterative development ⟶ Modularization ⟶ Software quality

Our participants (mainly the developers) reported that the iterative nature of Scrum allowed them control over their tasks and stronger focus on their quality because of the modularity of the tasks. They explained that when the Sprint goal is chunked into modular user stories, then this modularity bring about the quality of atomicity. Developers, in our sample, explained that atomicity gave them a sense of control, which made it easier for them to meet high standards in producing the output of the task. In addition, they experienced improved ability of better focus on the quality of the tasks.

Formal Inspection & Adaptation ⟶ Continuous feedback & frequent testing ⟶ Software quality

Formal Inspection & Adaptation Inspection and adaptation are tightly coupled in our data. Our participants implied that the outcome of inspection is adaptation. This is rooted in activities prescribed by Scrum. The guide states: "inspection enables adaptation. Inspection without adaptation is considered pointless" (Schwaber and Sutherland 2020, p. 4). Although inspection takes place informally when enabled by transparency, our data show that inspection and adaptation also have a formal basis carried out through prescribed ceremonies, such as in the Sprint reviews. The behavior induced by this process peculiarity is the frequent and timely feedback that the development team receives from the business users, either via the proxy of the product owner or, in some instances, the end users actively participate in the Scrum ceremonies. The feedback results in adaptations, such as actions taken by the development team to correct and align the increments with the business expectations. This process has a preventive impact on software quality.

Business users are given the opportunity to inspect the software increments before the release or even at the start of testing, preventing potential defects. Participant 19 explained how this process materialized in practice: *"because the development cycle is small so you can get feedback fast. Because development is small, the feedback is fast and therefore the improvement is fast. It goes with the process because if you don't improve, you end up making an application which is low quality. But if you learn and improve then the quality is much better; this is what I experienced ... Yes. And this continuous improvement is just because there is continuous feedback. It is not the same with waterfall. It is the process which actually enables us, because processes helps continuous development, this feedback is*

continuous improvement" (P19). Participant 31 asserted this conclusion further: *"the main strength [of Scrum] is the ability to get user feedback, and to use that to actually make the software do what it's supposed to do, and to make it function in the way that brings most value to the end user. So, they can say, okay, this is, this is technically correct what you've built, but it doesn't fit our workflow one hundred percent. So could we do these changes, and we'll get a better fit, we'll get less friction in using the system. So, I think that kind of ping pong with the user is very much helping increase quality in the solution in the end"* (P31).

Our analysis shows that while collaboration is mainly about helping each other's and contributing to the team's collective effort to produce a sound and high-quality product, formal inspection and adaptation on the other hand, is intended to correct deviations from the business needs. Instead, it has a corrective scope to align the developed features with the purpose of the software. In addition, while collaboration is carried out by informal and formal means, formal inspection and adaptation is mostly done in accordance with a process in a pre-defined ceremony. The business representatives carry out the inspection and provide feedback and point out any deviations.

$$\textbf{Formal Inspection \& Adaptation} \xrightarrow{v_6} \textbf{Continuous feedback \& frequent testing} \longrightarrow \textbf{Software quality}$$

Our participants accounts indicate that the this Scrum feature, i.e., formal inspection & adaptation, advances achieving software quality by fostering a process that generates continuous feedback from the end users. Subsequently, this feedback is invested to align the software features with the business needs. Our participants explained that frequent inspections in a short iterative development cycles, also mean more testing and rapid adjustments for better alignment of the software with the business needs.

4.3 RQ3: When Scrum Teams Fall Short of Improving Quality

With **RQ3**, we sought to understand what circumstances, conditions, and constraints cause the Scrum value-add that we identified in Phase 1 of the study to slow down or even fail to materialize. To this end, we studied two cases (**Kolkata** and **Phoenix**) of software development teams who used Scrum but failed to capitalize on its ability to advance achieving software quality. In this section, we report the findings of each case.

4.3.1 Kolkata

The project we studied in this case, was commissioned to develop and integrate a software platform for a US-based manufacturer of food ingredients, e.g., sweetness and protein fortification, for a variety of products, e.g., bakery, dairy beverages and animal nutrition. The scope of the newly developed platform was to build a workflow system for the validation and verification process of initially 52 products, then in future phases more products were planned to be rolled out to the platform. The development work also included the integration with seven existing systems, some of which were core systems for managing production and manufacturing processes. The existing systems had previously been supplied and maintained by different vendors. The client opted for custom development in lieu of an off-the-shelf product, due to the peculiarity of its processes and the unavailability of products oriented for this niche industry. The scope of the development also included a Mobile App for notifications and alerts to users.

The project was planned to last one year, including a three month period for user acceptance testing (UAT), which is the last stage of testing carried out by the end-users prior to signing off the roll-out to production. The client prepared a requirements document, which detailed specifications of functionality and workflows for the validation and verification of their products. The business rules were documented in a separate "product validation and verification specification" document for each product within the scope of the initial roll-out. **Kolkata** set up a team of 12 developers, six QAs, a "project manager", a "Tech Lead", a "senior business analyst" and a "QA Lead." Prior to the commencement of the development activities, the product owners, technicians, chemical engineers and other end-users from the client had several workshops with **Kolkata**'s "project manager", a "Tech Lead", a "senior business analyst" to transfer knowledge about the requirements.

Kolkata's team commenced the development in the first week of January 2020 using Scrum with one-week Sprint iterations. The team agreed with the client to demo the features developed at the end of every Sprint weekly; the Sprint concluded Friday of every week, and the demo took place every Monday. The team had well-established quality assurance processes and practices. They use pull-based development. Once a pull request (PR) was completed by a developer, it was made available for peer review. Developers were encouraged to perform unit tests prior to code review. Once a PR was reviewed and approved by a senior developer, it was merged with the upstream code base. In addition, the development team used a static analyzer (TSLint); developers were encouraged to run a code analysis of their code prior to unit testing. Once a Sprint release became available, it was moved to a testing environment for the QA team to commence "system testing."

The team had a well-structured infrastructure to support the Scrum process. It used a Microsoft Azure suite of tools to document, report and collaborate on artifacts, ceremonies (e.g., retrospectives actions) and planning (e.g., Sprints' user stories tracking). The company had adopted and used Scrum for slightly over two years for all its projects. Across the board, the company acknowledged that the implementation of the method was not optimal. We commenced our observation in November 2020; the team at that point of time started its 36th Sprint.[9]

By the end of January 2021, the system was not rolled out for production. Both parties extended the agreement by two months to "clean bugs." Our observations have ceased at this point. We learned later from our contact that the UAT period helped the developers to better understand the bugs. The interaction with the end-users allowed them to understand the deviations and the errors, and were supported in rectifying them by the end-users. Both parties decided to extend the project further to resolve bugs. Eventually, the system was released into production in May 2021 with "few bugs," according to the "senior business analyst" (our point of contact at the company). Five developers remained assigned to the contract for maintenance and further "bugs cleaning." We understand that a decision was made to roll out the software platform to production with "few minor bugs", presumably with little impact on the end-users.

Our observations showed that the team was producing releases with high defects density, the monthly defect report available in Azure Test Plan indicated that on average, each Sprint produced 10 "critical", 12 "medium" and 7 "minor" defects. Subsequently, Sprints were extended to reduce and resolve defects. The project was not delivered on schedule partly

[9]The count of the number of Sprints does not reflect the calendar dates. This due to holidays and delays experienced by the team. For example, some Sprints were extended by a week or more to conduct what the team called "bugs cleaning", a period dedicated to resolving outstanding defects in order to reduce the defect ratio.

due to the high number of defects. The end-users did not provide their sign-off of the system until all "critical" and "meduim" defects were resolved.

The analysis of observations data of **Kolkata** yielded three constraints, which hindered the ability of the team to achieve better quality. These constraints were: (1) inconsistency in the implementation of the prescribed Scrum process, its values and principles, (2) cultural constraints, and (3) inaccessibility to the end-users and business knowledge. These constraints hindered the Scrum value-add for quality to materialize. We used Phase 1 conclusions as analytical lenses to explain the deviations. This facilitated crisp thinking and the lenses led us to focus on a distinct interest, i.e., how these constraints manifested and hindered the Scrum value add to materialize.

While some of these constraints had their own unique origins, they created more constraints to exist and exasperate conditions further. The relations were intertwined, but they exerted a combined force on the team's ability to leverage the Scrum value-add to achieve better quality. For example, the inaccessibility to business knowledge resulted from a delicate pre-contract negotiation process. In order to secure the contract, **Kolkata's** sales team was less demanding. Thus they failed to negotiate access to the business knowledge and did not obtain the opportunity to involve a product owner in the Scrum team. This contributed to inconsistencies in the implementation of Scrum. Similarly, some cultural constraints such as fear of speaking up made the Stand-ups' inefficient, with developers being reluctant to talk about their impediments.

Inconsistency with the Prescribed Scrum Process Although most of the prescribed Scrum ceremonies were adhered to, their execution lacked rigor and openness. The daily Stand-ups were merely status updates from the developers. The Scrum guide explains that every Scrum event "is a formal opportunity to inspect and adapt Scrum artifacts" (Schwaber and Sutherland 2020, p. 7). In order to "inspect," the expectations are to "identify impediments" (Schwaber and Sutherland 2020, p. 9), then resolve them to ensure progress continuity, and improvements. Developers were usually not allowed to talk about their impediments, and when the opportunity was provided to express their concerns, they were still reluctant to do so. The Sprint planning was undemocratic. Developers had little say in the selection of their tasks, including their estimations. The "project manager", who usually assumed the role of the Scrum Master, or the "Tech lead" made the decisions regarding who will work on what and the estimate of the tasks. The Scrum guide states: "Sprint Planning initiates the Sprint by laying out the work to be performed for the Sprint. This resulting plan is created by the collaborative work of the entire Scrum Team" (Schwaber and Sutherland 2020, p. 8). This implies equal participation of all team members, which was not the case. The estimates made by the "project manager" or the "Tech lead" were usually inaccurate or too optimistic. The retrospectives were rather shy from meeting the intended outcome. While the Scrum guide advises that "the Scrum team identifies the most helpful changes to improve its effectiveness" (Schwaber and Sutherland 2020, p. 10), the team timidly mentioned the key problems, but no actions were taken to "improve its effectiveness" (Schwaber and Sutherland 2020, p. 8). Instead, the status quo was maintained, and issues experienced in previous Sprint cascaded onto future Sprints.

Some of the deviations highlighted above may be caused partly by cultural constraints. For example, the developers were reluctant to talk about their issues when they were given the opportunity. This could be a manifestation of power distance reported in some cultures. Hofstede's power distance suggests that less powerful team members in organizations accept that power is distributed unequally, thus influencing interpersonal interactions (Hofstede 1984, 2001). Hofstede's uncertainty (Hofstede 1984, 2001) avoidance could explain

the reluctance to talk about problems during the retrospectives. Cultures with a high degree of uncertainty avoidance view conflict as unpleasant and undesirable (Hofstede 1984, 2001). Oliver (2011) explains that uncertainty avoidance is more common when organizations prefer hierarchical structures in a power distance culture (Oliver 2011).

In addition to these cultural dimensions, the inconsistency with Scrum is also promoted by the effect of "falling between the cracks." The cascading effect of unresolved issues from Sprint to Sprint and the contractual promises with the client, i.e., demoing Sprint's features on a weekly basis, has resulted in pressuring the team to focus on producing features at the expense of fidelity to Scrum's guidelines, values, and principles. During an informal conversation between the researcher and the "project manager," the researcher asked about the reasons behind not asking and discussing impediments in the Stand-ups. He answered: *"most the time they [developers] don't want to. From experience whenever I ask nobody wants to say anything! Another reason, the team is large, we try to keep it to minimum ... You were in the retro the other day, we not meeting the goals, we try to save as much time as possible, so we cut on the ceremonies."* As per this account, the size of the team had also contributed to "falling between the cracks," i.e., adherence to Scrum is overlooked.

The QAs were not participating in the Scrum ceremonies. Only the "QA Lead"attended the Stand-ups. This fragmentation of the development team is also not in alignment with Scrum guidelines and Phase 1 reports. Our examination of the defects comment show that the QAs in some instances had different interpretation of the requirements. This misalignment of requirements understanding and separation of functions (QA and development) has furthered the decline of the effect of collaboration.

Some symptoms evidenced the decline of the software quality by Sprint 37. 121 defects were recorded in Azure Test Plan out of which 21% were assigned "critical" severity and 69% "medium." Developers were not given the opportunity to estimate their own tasks. Estimates suggested by the "project manager" or the "Tech lead" were overly optimistic. Subsequently, developers responded to this condition by "cutting corners." During Sprint 37 retrospective, two developers acknowledged: *"what did not go well in this sprint is we did not do enough testing. We messed a lot of scenarios. The testing has not been properly done."* The other developer seems to imply the failure to do any testing: *"this sprint we missed out on testing. We had to do a lot of testing."* Surprisingly, no action was taken, nor any follow-up was promised to address the issue in upcoming Sprints.

This constraint inhibited v_1, v_2, v_3, v_4 and v_6 to materialize. The unequal participation in the team impaired the behavior induced by collaboration. The Scrum values of "courage" and "openness" were not promoted by the project leaders, hence the effect of v_2 failed to take shape because team members were not encouraged to speak out. We learned from Phase 1 that when developers are empowered to estimate their tasks or user stories, they become accountable to meet the expectations. In addition, they cater for a quality buffer in the estimation of the task to assure the quality of their work. Accountability forges a desire to meet expectations, including quality. The behavior caused by this social antecedent did not materialize in the case of **Kolkata** because the adherence to a Scrum value, "holding each other accountable as professionals" (Schwaber and Sutherland 2020, p. 5), was disregarded. Consequently, v_3 has also failed to materialize. The visibility and access to the end-users was exclusive to the project leader, this made the project less transparent, i.e., v_4 effect did not take place. Finally, the end-users were not actively conducting inspections during the Sprints, this deprived the team from the effect of continuous feedback to learn and align the product with the business needs, hence v_6 effect did not take place.

Cultural Constraints As discussed above, some of the cultural constraints have hindered the implementation of Scrum. We observed that team members either did or did not do certain things because they culturally perceived the proper way to conduct oneself as per their own understandings and acted accordingly. It was apparent that developers were reluctant to voice their issues and obstacles during the Stand-ups and the retrospectives. When they did, their voices were evidently frail. When asked to provide "status updates", they usually kept it concise, then reclaimed their silence. This condition made most Scrum ceremonies inefficient and misaligned with the Scrum guidelines. The Scrum guide suggests, "The Scrum artifacts and the progress toward agreed goals must be inspected frequently and diligently to detect potentially undesirable variances or problems. To help with inspection, Scrum provides cadence in the form of its five events" (Schwaber and Sutherland 2020, p. 4). "Inspection" was not happening simply because team members were reluctant to speak up.

These cultural constraints are the fear to speak up during the ceremonies, avoiding speaking about problems during the retrospectives, and unilateral decisions made by the "project manager", relating to the developers' tasks, i.e., what to work on and the estimate to complete the work. As discussed above, these cultural constraints may have diminished the effect of some social antecedents; mainly, collaboration and psychological safety, v_1 and v_2. Participant 5, from Phase 1, explained bluntly the consequences of a culture that opts for silence instead of openness. He stated: *"if your culture is slide it under the carpet so no one else would know about that issue or that problem, then it's not agile fault, it is the fault of the people and the team and the culture. But if you have a team that is about communicating well, speaking well about each other, and addressing those proactively, not reactively, that's when you will be able to create the best software"* (P5). When people are culturally inclined not to speak up, then it becomes the role of the leadership to promote psychological safety to counter the effect of this cultural attitude. Participant 12 explained how his company communicated this value. He explained: *"and the company made it clear that no matter the consequences, obviously, except the illegal reasons, but no matter the consequences, people wouldn't be sanctioned for discussing their opinions, sharing their concerns and so forth. So, it was a very psychologically safe environment that people were encouraged and are really happy to work with their colleagues despite the distances between the offices"* (P12).

Inaccessibility to the End-Users and Business Knowledge The client documented the requirements in a "System Requirements Document (SRD)" for the **Kolkata** team as a source of truth for the expected system functionality. In addition, the team received 52 "product validation and verification specification" documents containing detailed products validation rules. Prior to the launch of the project and the commencement of the development activities, the client and three members of the **Kolkata** team, the "project manager," the "Tech lead," and a "senior business analyst," attended several workshops and requirements "debriefing" session with the client to transfer and familiarize the vendor with the requirements. These activities took place physically in the US. During an informal conversation between the researcher and a developer. The latter explained: *"the SRD explains in decent details the workflows. We also have product specification document for every product. They have the product rules, composition, compliance rules, testing procedures and all that but the language is very technical ... not easy to figure out."* The team relied on a secondary source of knowledge: the "project manager," "Tech lead," and the "senior business analyst." Understanding the requirements from a secondary source of knowledge were ineffective and insufficient. A senior developer conveyed to the researcher: *"they [the secondary source of knowledge] do not always know [the requirements] and sometimes getting answers from the client takes a long time."* He further explained that this condition results

in *"delays and obviously defects. Because we never sure."* This condition has also impacted the quality of frequent and direct interactions with the client in the process of inspection and adaptation. The "client demos" were conducted without the presence of the developers. Only the "project manager," "Tech lead," and the "senior business analyst" were present in these sessions. This deprived the developers of direct exchange of information and feedback with the system's real users and subsequently missed oout on knowledge exchange opportunities.

This condition contributed further to the decline of the software quality. An examination of the comments left by the QAs and the developers on Azure Test Plan defects showed a significant number of defects with comments like "waiting for clarifications from the client," "please, confirm with the client," and "ask [project manager, Tech lead or the Senior business analyst name]." This indicated that a significant number of defects were related to understanding the requirements - highly likely caused by assumptions made by the secondary sources of knowledge. At the end of each Stand-up, the "project manager" used to ask the developers with requirements questions to stay online to discuss and clarify their queries. In one particular instance, a developer stayed to ask the "project manager" questions. He had a specific question regarding the implementation of a feature, and a lengthy discussion took place. The discussion showed a considerable level of complexity of the product being developed. The developer seemed to experience difficulties while understanding the product requirements and the "project manager" had limited knowledge of the said requirements and lacked intimate details. The discussion without business expertise seemed ineffective. Most of the information exchanged was assumptions made about business rules and the expected system behaviors. The developer kept asking questions, and the "project manager" answers were mostly guesses; he was using a language showing mostly an inclination to make assumptions, e.g., *"I think this is what it means ..."*, *"let's try ..."* and *"have you looked at [product name] document?"*

The Scrum guide states: "Scrum Teams are cross-functional, meaning the members have all the skills necessary to create value each Sprint" (Schwaber and Sutherland 2020, p. 5). **Kolkata**'s team did not have the business "skills necessary" to understand the requirements better. In Phase 1, we learned that direct interaction with the product owner or business users facilitates the understanding of requirements and enables access to frequent feedback, which allows for fast adjustments so that software features are aligned with a more accurate presentation of the users' needs. This constraint deprived the team of an important asset, collaboration with the end-users, which has subsequently impacted their understanding of the requirements. As discussed above, to keep producing features and meeting the expectations of weekly Sprints releases and client demoing, developers made assumptions, which some were inaccurate, resulting in further defects. In comparison with Phase 1 conclusions, this constraint hindered the effects of v_1, v_4 and v_6. Participant 38, from Phase 1, explained that Scrum creates a one-team mindset, a quality that makes knowledge exchange efficient and fluid within the team. He explained: *"I mean, that in Agile, you have one team, and all knowledge is circulated within this team, and there is no need to pass this information. It's inside, it might be no formal documents, it might not be some website on Confluence, or wiki. And you store that information inside as a team, and there is not any information loss when you move further"* (P38). In the case of **Kolkata**'s team, the absence of the end-users in the team broke the "cohesive unit" described in the Scrum guide (Schwaber and Sutherland 2020, p. 5). The absence of the end-users (or even in the proxy of a product owner) denied the team access to frequent feedback, which is a type of knowledge with corrective implications. Participant 2 explained the consequences of not collaborating with the end-users: "very good question! Simply people do not collaborate and this mean more bugs

because the QA and the developers do not communicate efficiently. The developers make a lot of assumptions about the business requirements because they do not communicate efficiently with the end users and so on" (P2). In regards to v_4, the QAs and developers had no visibility into the clients feedback from the demos, and interactions. This has made the project less transparent and information communicated from the client has become exclusive to the project leaders only. This lack of openness also deprived the developers and the QAs from asking questions and knowing more. Participant 9 explained: *"well, if the project is transparent, it's bound to be more successful than the ones which are not. Because transparency leads to openness, transparency leads to more questions. More questions lead to more answers, and more answers leads to less bugs and better quality. So, the better the clarity that they have, it will reflect on the product"* (P9). v_6 was subject to a similar effect. The client demos were performed to showcase progress. They were not formal inspection exercises where developers and QAs could receive feedback and have "conversations" (P12, P30 & P32) to advance the team's understanding of the requirements, the errors, and how to fix them.

According to the assessment of the team itself, the business requirements were highly complex. A senior software developer stated to the researcher: *"we are not chemists you now! The business rules are so complex, nobody understands them and the documentation is difficult to get you head around."* The "project manager" concurred in a separate conversation. He said: *"we under-estimated the complexity from the beginning. It has impacted everything we do ... Now, the estimate not always accurate. The documentation we have is written for specialized audience, it doesn't always help us."* During Sprint 38 planning, while the "project manager" was assigning the tasks to the developers, a senior developer objected to some suggestions. Some developers apparently cannot take up tasks that are *"complex"* according to the senior developer. He was commenting hesitantly on those suggestions. His voice was frail and not showing confidence, and he could not complete his sentences. He would start talking, then stop abruptly. Developers were not given opportunities to comment on their assignments. The "project manager" continued assigning user stories while disregarding the objections by the senior developer. None of the other developers commented on his decision, nor were they given the opportunity to do so.

Complexity is not a constraint, as it does not contribute to the failure of the causal chains (v_1 ... v_6) promoted by the social antecedents. It is relevant to bring up in the findings, because it made **Kolkata's** team's need for access to end-users and business knowledge more noticeable. The complexity coupled with the inaccessibility to the business knowledge contributed to the shortcoming of meeting the Sprint goal and functionality not aligned with the business needs. During Sprint 37 retrospective, the "senior business analyst" had offered feedback. He commented: *"what went well is we managed to deliver our sprint even though we did not meet the scope 100%. At least we got something that we could demo."* Then, he said: *"what did not go well is regarding the spring goal. The sprint goal was extremely complex. I'm not disappointed at the team. They worked in a very complex functionality. During the demo, we found that the implementation of some business logic was a mess!"* Then he urged the team to make sure they ask questions from him or the "project manager" regarding the business logic. He stated: *"the next sprint will be a challenge. The team already putting extra hours and they may put more hours."*

The cocktail of these constraints has slowed down the potential of the Scrum value-add for achieving software quality, contrary to Phase 1 findings. The team ended up in a saga that was difficult to fix. Every Sprint has inherited the cascading effect of the previous Sprints that is further aggravated by the lack of inspection and adaptation of the process. In Sprint 37, the "project manager" negotiated with the client to extend the Sprint by one week that

was dedicated to "cleaning bugs." This endeavor did not yield a significant decrease in the volume of defects. Subsequently, both parties agreed to extend the "cleaning bugs" effort to Sprint 38. In addition to developing a tool for workflow management, the contractual agreement also included the development of a mobile App for users to receive and initiate action on system's alerts and notifications. It was decided that at Sprint 39, the development of a mobile App shall commence. The "project manager" decided to split the team; three developers were assigned with the task of "cleaning bugs," and the remaining eight developers were assigned to the development of the App. By the end of January 2021, the project deliverables were behind schedule and UAT did not start as planned. Azure Test Plan report showed 148 outstanding defects.

Table 10 summarizes how the constraints observed, in the case of **Kolkata**, clashes with the Scrum value-add for achieving software quality. Each cell in the table indicates whether a potential clash occurred as a result of the presence of the constraint in the development environment and the absence of a value, as identified and discussed in Phase 1. The clashes hindered the potential of the values to advance the ability of the **Kolkata**'s team to achieve better software quality. The symbol ✘indicates a potential clash, and **na** means that we have not reached a conclusion of a potential clash.

4.3.2 Phoenix

Phoenix entered into a contractual agreement to develop a Mobile App for a client based in Austin (Texas, USA). The product aims to establish a community of wine lovers for the Texas region who can share their feedback and knowledge of wine products. The other core feature of the App is to facilitate users to purchase wine either directly from the supplier or a third party while using the App as a platform to sell its products. The client is a digital solution provider. They design digital solutions and concepts for their clients, including the marketing and the rollout of the solution. Still, they do not perform the software development part. Instead, they commission it to other vendors. In the case of this product, it was **Phoenix**. **Phoenix** takes pride in becoming the *"go to place"* for developing Apps. However, this success is not all rosy. During the initial recruitment meeting with the CTO, he stated: *"we achieved a lot in the last 10 years ... I started developing Apps with my partner in my bedroom at my parent house. Now, our customers come to us because they like our Apps ... We get so much business that we can't handle ... I hate to see a review in the [App] store reporting a bug! It damages our reputation. Unfortunately, it has become frequent."*

The project was set to take five months, from November 2020 to March 2021. **Phoenix** set up a team comprised of five developers (one senior developer) and one UX designer.

Table 10 Constraints to the Scrum value-add observed in the case of **Kolkata**

Social antecedents	Inconsistency with Scrum	Cultural constraints	End users & knowledge inaccessibility
v_1—Collaboration	✘	✘	✘
v_2—Psychological safety	✘	✘	na
v_3—Accountability	✘	na	na
v_4—Transparency	✘	na	✘
v_5—Iterative development	na	na	na
v_6—Inspection & adaptation	✘	na	✘

The team received a "business requirements document (BRD)" as a source of requirements. Our examination of the document showed a great level of functionality details, including screens prototypes, color scheme specifications and detailed *"customer journeys."* The team commenced the development during the first week of November 2020 using Scrum with two weeks Sprint iterations. The client demos were conducted at the end of every Sprint. The team adhered to some of the conventional quality assurance practices, e.g., unit testing, system testing, and coding standards. The team had no dedicated QAs. Developers did unit testing and tested each other's developed features and end-to-end tests. The agreement with the client covered a UAT period where representatives of the client shall conduct testing of the App before signing off on the roll-out to production stage.

The team had a well-structured infrastructure with the tools to support its Scrum process. They use Atlassian products (e.g., Jira, Confluence and Bitbucket) to document, report and collaborate on artifacts, ceremonies (e.g., Sprint planning) and tasks progress (e.g., user stories tracking). The Scrum implementation was less conventional. For example, the "Team Huddles" were organized on an ad-hoc basis or whenever the CTO decided to need one. Conversely, Sprint planning scheduling was consistent. They were conducted every Monday of a second week. In this project, the CTO assumed the role of the Scrum Master. He explained to the researcher: *"we are a small place; we do not have the luxury of having a dedicated Scrum Master to every project ... So, I step in to do this role most the times."* A product owner or representative from the client was not part of the team. The development team relied on the BRD to understand the requirements. If further clarifications were to be sought, the CTO was the only point of contact with the client. He would seek additional information. Developers were not allowed to contact the client directly.

Overall, the project ran relatively smoothly. The difficulties were caused mainly by internal tensions. Most defects raised during UAT were induced by assumptions made by the developers due to gaps and unknowns in the requirements. The roll-out of the app to production was delayed by one month because of ten "high" priority defects with potential customer impacts. During the development, few defects were reported. Some defects were raised by the developers while testing each other's developed features. According to the senior developer, the client demos sessions were used for "showcasing" the progress. He further explained that the client's representatives were not given the opportunity to "inspect." Perhaps, this explains the defects not being identified early on in the process. Although we consider the number of defects reported relatively low, our findings show parallels with the **Kolkata** case. Therefore, comparing both cases will be valuable and helpful in drawing enriching and more substantial conclusions.

In the case of **Phoenix**, two main constraints have emerged during our analysis that have influenced the potential of Scrum value-add to materialize: team internal tensions and inaccessibility to the end-users and the business knowledge. Although, we observed some minor deviations in the execution of the Scrum process, they were symptoms of the two main constraints rather than a problem that warrants its dedicated category. For example, the retrospectives were shallow and lacking in openness. These are symptoms of the tensions in the team, not a breach of a Scrum recommendation.

Team Internal Tensions There was an apparent tension between the CTO and the rest of the team that was mainly caused by unaligned views regarding how Scrum should be leveraged and the uncompromising nature of the CTO leadership style. Our observations show that these circumstances caused the developers to become disengaged. Consequently, they became indifferent to the quality of their deliverables. This constraint compromised the

potential of v_1, v_2, and v_3. The effect of these causal chains failed to materialize because collaboration was inefficient, developers did not feel "safe" to speak up, and their inability to "influence" rendered them to become less accountable.

The senior developer described the CTO as a person with a style of *"my way or the highway"*, an American idiom which implies being assertive and portraying the view that there is no alternative apart from his or hers. A developer on the team described him as *"no bad news only good news."* He discussed: *"[the CTO first name] never likes to hear the bad news. He gets upset and agitated ... I'm always afraid to report anything he may not like. The other guys feel the same!"* When the researcher asked about the consequences of this state, he replied: *"we have to get it right all the time, whether we understand the requirements or not ... We never have direct contact with the clients. It is not possible ... So, we usually assume and carry on to avoid bad news."* This state did not only cause the developers to become resentful of their leader, but it also had far-reaching consequences on the quality of the product. Most of the defects reported in UAT were either caused by inaccurate assumptions or the indifferent attitude of the developers. For example, a particular defect could have been prevented if the developers were empowered to report "bad news" while feeling "safe" to do so. This particular defect was in relation to the wine products databases used by the App. The client expected that the databases should have a process in place to be refreshed each time when a new version from the databases becomes available. Although this was not specified in the BRD, direct collaboration with the client or feeling "safe" to report this gap in the requirements could have prevented the defect. The researcher asked the senior developer to comment on the causes of the defect. He explained: *"I was almost sure, it did not sound right. You can't use the same versions [of the databases] all the time. Products comes and go ... But honestly, I didn't bother to ask. To me, if it is not in the BRD I don't care."*

This constraint compromised v_1, v_2, and v_3 of the Scrum value-add to materialize. The testimony of the senior developer quoted above demonstrates the inefficient collaboration. Phase 1 participants reported "working well together," and the Scrum guide describes teams as "cohesive units." In the case of **Phoenix**, the developers were not working well with their CTO. The leadership style created a state of disquiet when speaking up because the CTO wants to hear only "good news." We learned from Phase 1 that feeling "safe" in a Scrum environment encourages the quality of care amongst the team. In **Phoenix**, this was not the case, e.g., *"I didn't bother to ask."* The senior developer statement also hints at the lack of accountability; although he was aware of the gap in the requirements, he did not feel any obligation to report it. This was aggravated by fear of speaking up and inefficient collaboration.

Compared to the **Kolkata** case, these tensions have parallels with the cultural constraints. This may imply that Scrum implementations should be supported culturally. Participant 2, from Phase 1 explained: *"If you don't have the values and the culture right from the start, you don't have a solid foundation. You'll be in the building, you need to put a solid foundation first. And that is the culture, the values. When it comes on the top of the building, those stores, those are the frameworks. If you put the Scrum and there on that very weak foundation. It's going to crumble. And it's not going to produce good quality software"* (P2). Scrum does not "work in vacuum"; It operates in a cultural and social context. **Kolkata** and **Phoenix** cultural peculiarities have hindered some attributes of the Scrum value-add.

Inaccessibility to End-Users and the Business Knowledge Similar to the **Kolkata** case, **Phoenix** developers did not have access to the client. Only the senior developer was attending the "demos" with the CTO. This deprived the team of frequent feedback (v_1, v_4 and

v_6) and a better understanding of the requirements (v_1). As discussed above, assumptions were made, which subsequently caused defects. During the first Spring retrospective, the UX designer raised an issue and suggested to the CTO to work on it. He reported that the UI designs required *"fast review and feedback from the client to meet the Sprint goals ... It would have been better if we were able to talk to [the name of the client]."* The CTO, then recorded "lack of communication with the client" as an issue to resolve and the action to take was "review project documentations." This was a clear signal to the team that they would have to rely solely on the documentation provided by the client. The visibility into the client's interactions and access to the end-users feedback was also exclusive to the CTO, this has impacted the effect of v_4 from developing.

Table 11 summarizes the constraints we found in the case of **Phoenix**. Each cell represents a clash (i.e., the symbol ✘) with a particular Scrum value-add for achieving software quality. When we have not concluded a potential clash, then we used **na** to indicate so.

4.4 Integration: the Scrum Value-Add and the Potential Constraints

To sync the various concepts, identified in response to **RQ2** and **RQ3**, in this section, we propose to integrate the findings of Phase 1 and 2 to propose a consolidated model to paint a holistic picture of the observed phenomenon. The model is a conceptual representation of when the Scrum value-add for software quality transpires and what constraints impede its potentials. Figure 4 depicts this synthesis. For simplicity, we used the social antecedents to represent the causal chains (v_1 ... v_6), instead of a fully-fledged graphical representation, as in Fig. 2. We use N in the diagram to make reference to the number of mentions in total in the data of Phase 1 for a particular causal chain. The purpose of doing so is to emphasize the significance of each of the chains in promoting the Scrum value-add. In the proposed model, we suggest failure points (presented in the diagram in the form of yellow arrows down callouts). These are points of failure in the causal chains of the Scrum value-add.

The causal chains promoted by the social antecedents (v_1 ... v_6) are in light gray rounded boxes. Each causal chain has failure points represented by the presented by the amber down arrow callouts are the constraints observed in **Kolkata** and **Phoenix**. The red circles show a potential slow down of the effect of the causal chain.

Collaboration and its effect (v_1) is a prominent (N = 30) feature of the Scrum value-add for advancing software quality. At the same time, it is jeopardized by more constraints compared to the other social antecedents. Collaboration fails to promotes its effects when three constraints arise in the Scrum environments: the implementation of Scrum is inconsistent with the Scrum guide recommendations, cultural constraints such as the ones observed in **Kolkata** leading to a fear to speak up and to avoidance of problems, team tensions caused

Table 11 Constraints to the Scrum value-add observed in the case of **Phoenix**

Social antecedents	Team internal tensions	End users & knowledge inaccessibility
v_1—Collaboration	✘	✘
v_2—Psychological safety	✘	na
v_3—Accountability	✘	na
v_4—Transparency	na	✘
v_5—Iterative development	na	na
v_6—Inspection & adaptation	na	✘

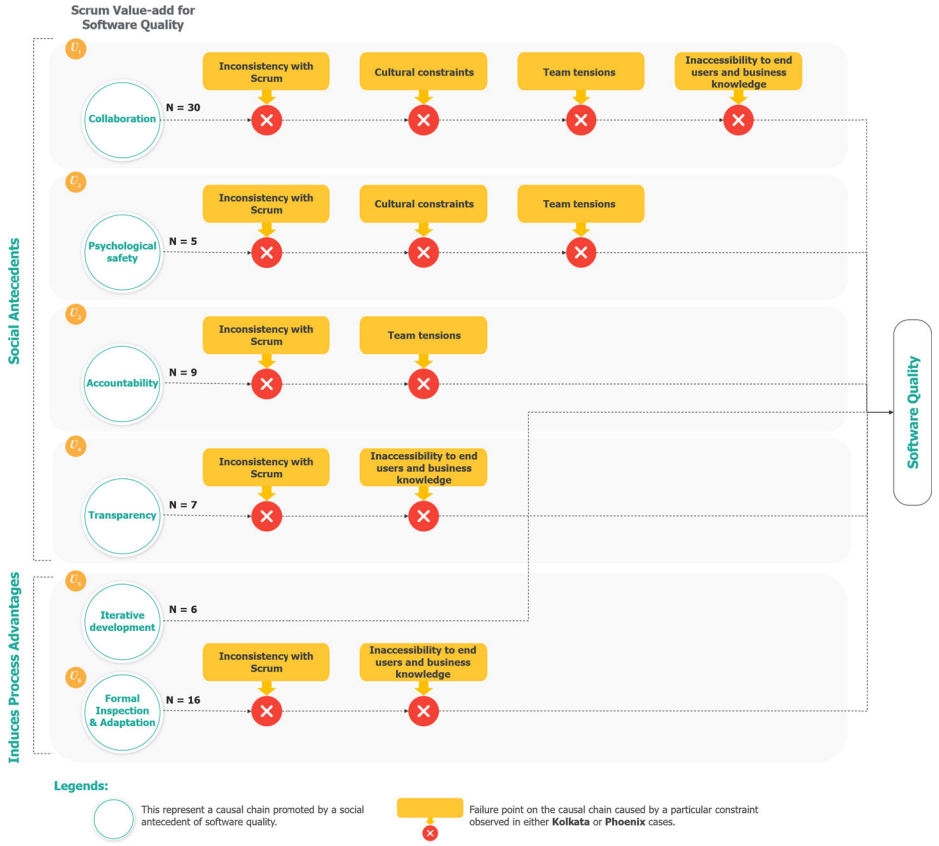

Fig. 4 Scrum Value-add and the constraints causing failure points

by clashes of expectations and inaccessibility to the business users and knowledge. While this social antecedent received more acclamation compared to the others, it has a high level of vulnerability, i.e., more failure points. This is an indication that this antecedent should be safeguarded and nurtured meticulously.

Although psychological safety was not as prominent in the data as collaboration, our participants emphasized its importance in order for some Scrum values to arise. This became evident in Phase 2 cases. The presence of this antecedent in **Kolkata** and **Phoenix** could have eventually mitigated the effect of some of the constraints and helped the teams to become more open about their own impediments. This causal chain related to this antecedent is potentially compromised by three constraints: (1) inconsistencies in the use of Scrum in regards to the promotion of the Scrum values of "courage" and "openess", (2) cultural particularities of **Kolkata'** work place and (3) the tensions caused primarily by the leadership style of **Phoenix**. In Phase 1, some of our participants (P3 and P12) indicated that this antecedent can be facilitated by leadership initiatives, e.g., *"... The company made it clear ... it was a very psychologically safe environment"* (P12).

Both transparency and accountability and their effects have slightly different weightings in our Phase 1 data (N = 7 and N = 9). In the case of **Kolkata**, accountability and its

effects did not materialize simply because it was not followed as per the recommendations of the Scrum guide. The team did not show a shared collective accountability. In the case of **Phoenix**, the team tensions have caused the team to become less caring, i.e., the wanting to meet quality expectations was not observed. Similarly, transparency was not adhered to by both teams, because of the exclusivity of access to the client and end-users has made the environment less transparent.

Formal inspection and adaptation is the second most cited causal chain in our Phase 1 data (N = 16). Even though it is relatively easy to implement, both cases failed to do so. This is a breach of a fundamental Scrum recommendation worsened by the inaccessibility to the end-users. Both Phase 2 cases missed the opportunity to interact with the end-users and receive continuous feedback. Subsequently, the effect of adjusting the product to the business needs earlier in the process did not materialize and defects were found in UAT phases.

In Phase 1, our participants reported that the iterative approach of Scrum permits modularization of tasks. This atomicity allows developers to have control over the tasks and focus on their quality. **Kokata** and **Phoenix** teams adopted the iterative recommendations of Scrum using Sprints. In a two separate conversations with developers from both teams, they confirmed to the researcher that they support Phase 1 participants claims. They stated: *" ... of course it [iterative development] helps. The quality of the code is better and easy to review ... But as we discussed, all the defects are related to requirements. You can write good code but functionally it can still have errors"* (**Kolkata** developer) and *"Yes, of course. Over the years, it allowed us to reuse many components and libraries. Because the functionality of Apps we get have similarities"* (**Phoenix** developer). This indicates that this process advantage enables better code and design (i.e., internal quality) and not necessarily the external quality. Hence, in the diagram, the iterative development causal chain does not have any failure points.

5 Validating the Findings

We used feedback from participants to scrutinize the validity of our conclusions. Feedback from participants is a technique for the researchers to validate the accuracy of their interpretations by allowing participants the opportunity to confirm or deny the accuracy of the conclusions made from the data, thus adding credibility to the qualitative study (Creswell and Miller 2000). This exercise is carried out once the analytic categories, interpretations and conclusions become available (Miles et al. 2014).

We implemented this validation process using two separate focus groups. We used a similar approach in a previous study (Alami and Paasivaara 2021) successfully. This approach allowed us to have access to direct feedback from our participants in an efficient and interactive manner; we had the opportunity to explain our interpretations, discuss them, and obtain their feedback. It also allowed us to fine-tune our conclusions. We decided to organize two separate focus groups, the first with a diverse representation of various roles (e.g., management, Scrum Master and QAs) in our sample and the second group with developers. We chose to have a dedicated session with developers, given the key role that developers play both in traditional quality assurance processes and in the causal paths uncovered in our study. We invited all our interviewees to participate in this validation exercise. Eleven participants volunteered to contribute, six in the mixed group and five in the developer group. The first focus group involved P2, P3, P12, P16, P18, & P22 and the second group P30, P31, P36, P37, & P38.

During the focus groups, participants were presented with our interpretations of the data and were 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 reflected their views and experiences. The focus groups participants were supportive of the findings. There was no categorical objection to the findings presented for validation. Most comments we received highlighted the importance of the "cultural fit" (both focus groups), "real end-user participation" (both focus groups) in the Scrum team with the relevant "expertise" (both focus groups). The slides and transcripts of the focus group sessions are available here.[10]

6 Discussion

In this section, we first discuss how our findings on social antecedents and process-induced advantages relate to previous work. Then, we draw some recommendations for practice based on our empirical data and understanding of how some Phase 1 participants managed to materialize the Scrum value-add for software quality. Mindful of Phase 2 cases, we tuned our recommendations to prevent the failure points (discussed in Section 4.4) caused by the different constraints inherent to the cultural and organizational particularities of **Kolkata** and **Phoenix**.

The social antecedents for software quality we identified in this study point to several established concepts in social sciences, information systems, and software engineering. However, to the best of our knowledge, the effects of these concepts on software quality have not been established previously to the extent of our work. With the exception of Prechelt et al. (2016), most of the available literature focused on establishing evidence for the efficiency of agile methods in improving software quality, with little attention to how these methods bring value to the teams' efforts in achieving quality.

Although issues of *collaboration* have attracted strong research interest from the 1980s (Kraus and Kraus 1980) until today (Bjarnason et al. 2022), surprisingly few studies have explored the specific effect of collaboration on software quality and the conditions under which these effects break down. Among the few exceptions are Çaglayan and Bener (2016), who investigated the association between the number of collaborators in specific code changes and the likelihood of introducing new defects through the code change. They found that the likelihood of introducing new defects increased with the number of collaborators involved in a code change (Çaglayan and Bener 2016). While this finding focuses on the number of collaborators and its potential adverse effects on software quality, our findings show how qualities of collaboration (e.g., collaborative activities related to knowledge sharing, feedback and learning) can benefit quality. In the context of agile, Hoda et al. studied the impact of inadequate customer collaboration in self-organizing agile teams (Hoda et al. 2011). Their findings suggest that inadequate customer involvement in agile projects can produce unfavorable effects that jeopardize project success. They reported that the nature of a fixed-bid contract pressured the team to over-commit, which resulted in issues in the elicitation, clarification, and prioritization of requirements, issues in access to feedback, loss of productivity, and even loss in business opportunities (Hoda et al. 2011). The key role of user feedback is also highlighted in Prechelt et al.'s grounded-theory study (2016). They found that direct, realistic, and quick feedback is important for defining accurate requirements and achieving high quality. Hanssen and Fægri highlighted not only the benefits but

[10] https://doi.org/10.5281/zenodo.6624063

also the cost of customer participation in agile teams (2006). They found that customer collaboration in agile projects motivates developers, increases confidence in prioritization decisions, improves communication and understanding of the end users' real problems, and increases visibility into the organization's internal processes (Hanssen and Fægri 2006). At the same time, their study shows that customers can also be unpredictable collaborators and lack continuity and relevant expertise (Hanssen and Fægri 2006). Our findings concur with these prior studies (Hoda et al. 2011; Prechelt et al. 2016; Hanssen and Fægri 2006) on the important role of user feedback. However, our findings go beyond this work by showing the breadth of mechanisms in which collaboration in Scrum teams contributes to quality. These mechanisms include not only user feedback, which benefits external quality, but also peer-to-peer learning among software engineers, which is important for internal quality. Moreover, while the mentioned studies identify contracts and user availability as conditions that may jeopardize collaboration, our study adds that cultural constraints, team tensions, and inconsistent Scrum implementations may, too, compromise the potential quality benefits associated with the use of Scrum collaboration processes.

Psychological safety appeared in organizational studies in the mid-1990s (Edmondson 1999a), but its roots date back to the 60's (Schein and Bennis 1965). Over the last decade, studies have presented mounting evidence of its positive effects on learning (e.g., Tucker et al. 2007), performance (e.g., Edmondson 1999b), employee engagement (e.g., May et al. 2004) and teams relationship (e.g., Gibson and Gibbs 2006). For example, Gibson and Gibbs examined the effect of a psychologically safe working climate on geographically distributed teams (2006). They found that a psychologically safe communication climate helps mitigate the challenges inherent to working in a dispersed working environment. Team members overcome the challenges because they are encouraged to speak up, seek help, and work out their issues constructively (Gibson and Gibbs 2006). Although reports from Google point to the benefits of psychological safety in software development contexts (Duhigg 2016), psychological safety has thus far attracted relatively little interest in software engineering studies. Conceptual work has recently advocated for a stronger focus on psychological safety in agile software development (Hennel and Rosenkranz 2021), but empirical work on this issue, including the effect of psychological safety, is difficult to find. Against this backdrop, one important contribution of our work is to unveil that psychological safety helps create an environment in which team members dare to care about quality by speaking out and by making the required efforts for quality. Another important contribution is that we point out that these benefits may be weakened by inconsistent Scrum implementations, cultural constraints, and team tensions.

Accountability is a core principle underlying social and organizational behavior in the effective governance of projects (Müller 2010; ul Musawir et al. 2020). Accountability relationships develop in groups between an actor (e.g., Scrum team member) and a group (e.g., Scrum team) when a responsibility is delegated to an actor and the group makes the actors accountable for the execution of the responsibility (Bovens 2007; Messner 2009). It influences organizational and individual behavior (Roberts 1991) and has positive effects on task performance (Yarnold et al. 1988). In an accountability relationship, an obligation from the actor to justify her conduct becomes a norm and the group is entitled to ask questions about performance, and make judgments about the actor and her conduct, with corresponding consequences, including rewards and sanctions (Burga et al. 2021). Methods to hold an actor accountable include reporting performance and the consequences associated with meeting or not meeting expectations (Burga et al. 2021). This social process takes also place in the context of achieving software quality in Scrum teams. After backlog items are assigned to developers, developers are accountable for instilling quality in their

code contributions and for meeting Sprint goals, for which they are held accountable in the Spring review (Schwaber and Sutherland 2020). While Prechelt et al. (2016) find that the absence of testers helps increase developers' accountability for the quality of the code they produce, our study shows that Scrum promotes accountability even in teams with dedicated testers. More specifically, the self-governing nature of Scrum teams and their strong emphasis on collaboration intensifies social processes through which individuals take personal pride of producing code that lives up with high standards of technical integrity and develop a strong desire to meet their team members' expectations for quality. In line with the idea that accountability emerges both from Scrum rituals and from social processes, we find that inconsistencies in Scrum implementations and team tensions may compromise its benefits for software quality.

Previous studies have reported that agile methods enhance projects' *transparency* within the team (Chong 2005) and enhance trust (McHugh et al. 2011). When contributions are not transparent, individuals may act selfishly (Bag and Pepito 2011; Eisenhardt 1989). However, when contributions are transparent, individuals invest higher efforts (Bag and Pepito 2011; Eisenhardt 1989). Transparency induces pressure, which may lead to higher efficiency. Menusch et al. suggest that when actions are unknown, a stigmatization effect, and more peer punishment is received (2014). When full transparency of actions arises, it creates full accountability because it allows targeted punishment and subsequently increases contributions (Khadjavi et al. 2014; Eisenhardt 1989). Against this backdrop, a key insight from our study is that, beyond its effect of incentivizing stronger efforts, transparency also promotes voluntary inspections, which stimulate feedback processes and thus enhance software quality.

Iterative development has been praised for contributing to software quality. The continuous feedback in the iterations enables responsiveness and learning from defects found in previous iterations leading to quality improvement (Jalote and Agrawal 2005; Krancher et al. 2018). However, strong evidence of its benefits for achieving higher quality is still lacking. Our findings suggest that, beyond its role in promoting feedback, iterative development also promotes modularization, forcing teams and developers to identify small self-contained units of work for a given sprint. Well-defined units instead of big, uncontrollable chunks allow developers to have control over the code and better focus on quality. Prior research has shown that modularity is an important determinant of software quality (English et al. 2016). It promotes evolvability, changeability, and maintainability amongst other facets of quality. In a similar vein, Prechelt et al. (2016) found that modularity is an important prerequisite for leveraging the benefits of Scrum. While extensive work has thus shown the benefits of modularity, an important insight offered by our study is that iterative development triggers developers to invest efforts in the modularization of designs and code.

Previous studies related to our theme of *formal inspection and adaptation* have focused on retrospectives and sprint reviews. Hassani-Alaoui et al. found that retrospectives can lead to process improvements and, hence, increase project success and customer satisfaction (2020). Unfortunately, this practice is not always sustained. They also found that, in some cases, senior management perceive retrospectives as a "waste of time" which leads to resentment from the team (Hassani-Alaoui et al. 2020). Dysfunctional retrospectives are not effective either in yielding improvements (Hassani-Alaoui et al. 2020; Eloranta et al. 2016). Eloranta et al. suggest that long Sprints (4 weeks or more) delay the feedback that is provided through sprint reviews (2016). The longer the Sprint, the higher is the likelihood of developing features that are misaligned with business needs (Eloranta et al. 2016). Delays in producing feedback for the development team can cause significant rework and process overload (Eloranta et al. 2016). So far, evidence has emerged from previous work to

show the effectiveness of adopting formal inspection and adaptation but with little insights into the effects on the product. Our work establishes this effect by showing that adherence to this Scrum recommendation yields continuous feedback and frequent testing, which contribute to achieving software quality. We also provide evidence of conditions under which this effect fails to materialize. In both Phase 2 cases, the "client demos" were used to showcase features without active inspections from the end-users. This proved ineffective because simply showcasing is not an active inspection.

It has become apparent from this discussion that our proposed social antecedents and process-induced advantages for software quality have either dualistic or even multi-dimensional relationships. For example, psychological safety promotes accountability and so does transparency, but in different ways. While psychological safety establishes a shared sense of responsibility, transparency creates visibility and access to information to ascertain whether individuals adhere to pursuing team's expectations. Similarly, inspection and adaptation are "pointless" without effective collaboration. The objectives of our work is to establish the evidence of the causalities of the social antecedents and process-induced advantages effects of software quality. As we will discuss in Section 6.2, future work could focus on interactions between these causal paths.

6.1 Recommendations for Practice

Table 12 documents the recommendations we draw from the data. The first column is a reference to the social antecedents' causal chains (v_1 ... v_6). The ownership is the party that we recommend being responsible for implementing the actions. For some recommendations, ownership is collective (e.g., team), indicating that the actions should be owned by the group. Leadership includes senior management, middle management, and team leaders. Some Phase 1 participants emphasized the need for organizational initiatives to be adopted by all levels and cascaded down the hierarchies efficiently. The recommendations are actionable measures to implement in a Scrum environment to promote the Scrum value-add for achieving software quality. These recommendations are anchored in our data, with the last column showing specific sources of the recommendations in Phase 1 and 2 data. Some recommendations are drawn from the analysis to integrate **RQ2** and **RQ3**, e.g., encourage reporting "bad news", not only "good news".

Our recommendations are unlikely to have high economic costs. They may not require major organizational transformations but rather incremental organizational change and team development, which can happen by considering these recommendations as targets to achieve. These recommendations are not exact strategies on how to operationalize the Scrum value-add for software quality. Our aim is to provide insight into what Phase 1 participants have experienced and proposed as potential measures to consider. However, they are granular and straightforward enough for leaders and teams to operationalize them based on their particularities and preferences. It is important to note that these recommendations are derived from data stemming from diverse contexts. For example, P3 worked for a large American software development vendor with teams dispersed across the globe, and so did P16 and P22. On the other hands, P2 and P18 worked for small-sized software suppliers with local clients (P2) and external clients in Europe (P18). In addition, some of the developers in our sample had over 15 years of experience (e.g., P31, P37 & P38) and had worked in teams with various levels of performance, efficiencies and abilities to deliver software quality. This diversity raises our confidence in the transferability of these recommendations.

Table 12 Recommendations for scrum software development implementations

Social antecedent	Ownership	Recommendations	Reference in the data
v_1—Collaboration	Scrum team	Pay attention to equal participation	P18, P36 & **Kolkata**
		Engage all participants in the Scrum team as early as possible on the process	
		Provide equal opportunities to contribute	
		Provide equal opportunities to speak up	
	Team members	Embrace and practice a collaborative mindset	P18, P22, P2, P16 & P31
		Embrace a team mindset	
		Practice constructive communication	
		Exercise empathy toward other team members	
		Exercise openness	
		Be open to feedback and provide constructive criticism	
	Business stakeholders	Ensure commitment and support for the Scrum team	P16, P12, P22, P30, P31, P37, & P38
		Participate in the Scrum team with competent and expert end users	
		Ensure consistent participation in the Scrum ceremonies	
		Share knowledge socially with the Scrum team (e.g., face to face)	
v_2—Psychological safety	Leadership	Establish and maintain a psychologically safe work climate	P12, P3, P16, **Kolkata** & **Phoenix**
		Promote a culture in which mistakes are acceptable	
		Promote openness	
		Recognize courageous acts of speaking up	
		Appreciate and acknowledge the behavior of showing vulnerability when offering new ideas	
		Appreciate and acknowledge the behavior of asking questions and sharing mistakes	
		Encourage reporting “bad news”, not only “good new”	
v_3—Accountability	Leadership	Empower the team to make decisions on their on work	P12, P16, & P38
		Reduce interferences with the team organization and decisions	
		Create leeway for the team to become self-governed	
v_4—Transparency	Leadership	Share decisions & information equally	**Kolkata** & **Phoenix**
		Implement and maintain a transparent infrastructure for information sharing, process artifacts and team decisions	
v_5—Iterative development	Scrum team	Write concise and manageable user stories	P2, P12, P16, P18, P36 & P38
		Engage and obtain the developers approval of the Sprint’s scope and goals	
		Encourage the developers to take ownership of the user stories estimation	

Table 12 (continued)

Social antecedent	Ownership	Recommendations	Reference in the data
v_6—Inspection & adaptation	Leadership	Ensure inspections by end users take place and not simply "showcasing" features at the end of each Sprint	**Kolkata** & **Phoenix**
	Business stakeholders	Actively test and provide feedback as soon as possible Commit end users to conduct deep inspections (i.e., "showcasing" are not inspections)	P12 & P16

6.2 Future Work

During the execution of this study, we became aware of potential future work to either extend some of the findings, broaden the empirical coverage, or further validate our concepts. Below we have listed some potential future work.

- **Interactions between social antecedents.** Our work focused on identifying causal paths induced by Scrum values, principles and practices to improve a team's ability to achieve quality. However, our analysis did not investigate the way in which each of the social antecedent is related to other antecedents. Although we establish empirical evidence of the causality of the different social antecedents and process-induced advantages, the interactions between these factors remain an area to explore. This would allow us a better understanding of the interdependencies and the impetus of these social antecedents in relation to each others. For example, during the first focus group, all participants in the group suggested that transparency is not a *"stand alone"* (P3) variable; it works in conjunction with the other antecedents. Participant 2 emphasized this: *"I don't think they [antecedents] work in isolation"* (P2).
- **The relationship between organizational culture and the Scrum value-add for quality.** While our findings on Phase 2 point to a role of national culture, our study does not focus on the role that organizational culture may play in strengthening or inhibiting the causal paths uncovered in this study. Yet *"cultural fit"* (P12, P3, P22 & P18) was highly talked about during the first focus group of the validation exercise, suggesting that other facets of culture beyond national culture are worth exploring. Prior work suggests that issues of culture, and mindset play important roles in agile transformation (Hoda and Noble 2017; Alami and Paasivaara 2021). For instance, our previous work (Alami and Paasivaara 2021), shows that technical excellence depends on an "agile mindset." This was also discussed during the feedback session with some participants. Future work may consider examining the relation between the ability of an organization to adhere to agile values (having an "agile mindset") and the potential of the Scrum value-add for quality.
- **Collaboration with business users.** Our findings emphasize the benefits of collaboration with the business users within a Scrum team. However, prior work has shown that it may sometimes be difficult to fully involve business users (Hoda et al. 2011) and that continuous involvement of business users can also compromise quality (Krancher 2020). In line with this work, we were cautioned during the feedback session that intensive collaboration with business users is not always the case. Instead, a *"true representation of the end users"* (P16 & P12) is the ideal collaboration with the business.

Future work could look at various levels or forms of business users participation in Scrum teams and their effect on contributing to achieving quality. This will help clarify under what circumstances the potential of v_1 can be most strongly leveraged.

- **Antecedents of internal versus external quality.** Although the distinction between internal and external quality emerged from our analysis regarding **RQ1**, our analysis regarding **RQ2** did not allow us to separate causal paths for internal quality from those for external quality. Our analysis did not allow this because we did not ask specifically about antecedents for external vs. internal quality, given that this distinction emerged only from our analysis. Future research could, hence, explore to what extent different social antecedents and process-induced advantages contribute to internal versus external quality. One could, for instance, test the idea that collaboration with customers contributes to external quality while factors internal to the development team, such as accountability and psychological safety, contribute to internal quality. Such an analysis may help explain why certain quality dimensions turn out to be more in focus in one case than in other cases, as in the cases of **Phoenix** and **Kolkata** where issues of internal quality rarely surfaced in the data.
- **The interplay between traditional quality assurance practices and the Scrum value-add.** Although the social antecedents for software quality may not have a tangible influence on quality, like other quality assurance tools and practices such as static analyzers or user acceptance testing that help point out deviations for coding standards and functional requirements, they advance achieving quality socially. Still, this social effect was described, for example, by participant 31 as *"powerful"*, and participant 2 experienced noticeable results, *"I have seen it in practice, developers write better code when they feel accountable"* (P2). Nonetheless, there could be an interplay between both streams (traditional quality assurance practices and the Scrum value-add). For example, we observed that **Phoenix** developers did not report a gap in the requirements and preferred to wait until UAT for the client to report a defect; this is because they did not feel obliged to do so. Would similar behavior take place for the adoption of a quality assurance process? I.e., developers would not use a static analyzer tool because they do not feel obliged to do so. This interplay could unveil the power of these streams of quality, i.e. Scrum value-add, and the traditional quality assurance practices. This could be investigated in future work.
- **Gender differences in attitudes towards software quality.** Future work could also examine the importance of these social antecedents and process-induced advantages from a gender perspective. For example, our work did not look at whether males value accountability more than females. The same could apply to psychological safety, i.e., do women value safety more than men? And to what extent do these antecedents influence men's and women's behaviors towards achieving better software quality?

7 Limitations & Assumptions

We identified some methodological limitations, which we discuss below.

- **Representation of the data.** The constraints of Scrum value-add for achieving software quality identified in phase 2 of the study are contextual to the two cases, **Kolkata** and **Phoenix**. The narrow focus of this methodological choice has some inherent limitations. Other constraints (e.g., organizational culture) may exist but were not salient in both cases and, hence, did not make it into our integrated model proposed in Fig. 4. This

may also apply to Phase 1 data. Although we consider our sample large and diverse, the antecedents yielded by the data may not include all possibilities, i.e., other antecedents could exist. This does not undermine our findings. On the contrary, we proposed conclusions based on extensive data and humbly believe it is a starting point for future work to broaden some of these findings empirically. For example, a quantitative instrument such as a survey could be used to capture other constraints and expand our proposed model.

- **Validation of RQ2 findings.** We had only 11 participants for Phase 1 focus groups to validate **RQ2** findings. This was mainly due to logistical issues; we were not able to organize further focus groups due to the geographic distribution and availability of the participants. From previous experiences, e.g., Alami and Wąsowski (2019) and Alami et al. (2020), sending written interpretations and asking participants to comment is not as efficient as an interactive session. For example, abstract concepts are not easily understood by participants because they do not relate to them. Conversely, interactive sessions allowed us to explain our concepts and to exchange additional details.
- **Validation of RQ3 findings.** We did not formally validate the findings of **RQ3**. However, we were asked by both companies to provide feedback periodically and at the end of the observations. We used these feedback sessions to discuss our interpretations throughout the study and at the end. In addition, informal conversations were used throughout Phase 2 to validate the researcher's interpretations and support them with additional empirical data.
- **Measurement of software quality in Phase 2.** We could only make conclusions based on observations and reports from **Kolkata's** and **Phoenix** team members. We did not have access to the software code or the final product to make further assessments of its quality. However, making conclusions based on defects reports and informants' accounts is a reliable source of empirical data. Having access to the actual software and its code may have yielded additional findings. For example, we could have been able to assess the quality of the code (i.e., internal quality).
- **Visibility to the client.** Our agreements with both cases, **Kolkata** and **Phoenix**, clearly stipulated no access to the clients, the end-users and no participation of visibility into the communication or interactions. This choice was made by both companies. This condition has prevented us from having the clients' perspectives. Hence, our Phase 2 data may be skewed towards the vendors' perspectives. Although the study would have benefited from access to client informants in the two cases, we believe that potential bias is attenuated by the representation of product owners and management in Phase 1 data.
- **Additional variables not included in the analysis.** Our **RQ2** and **RQ3** are more concerned with the effects of Scrum as a value system on achieving software quality and how they fail to materialize. This crispy focus limited our scope to include further variables in the analysis, such as gender, organizational particularities and technologies used to develop the software (e.g., programming languages, development frameworks, etc.).

Our analysis and interpretations have some assumptions. Below, we disclose the assumptions we have become aware of during the course of this study.

- **Case studies observations time frame.** We assumed that three months of observation were sufficient to investigate **RQ3**. This duration is similar to what other software engineering researchers have used in their observational research (Sharp et al. 2016). Yet a longer period may have yielded further findings. Future work may consider longitudinal studies on this topic.

- **Project management decisions.** We assumed that project management decisions had no influence on the conditions we observed. For example, in the **Kolkata** case, the decision to execute the project in a one-year time frame may have placed further pressure on the team. The contractual agreement and promises made to the client may have given the rise to additional constraints (Hoda et al. 2011). For example, in both cases, the companies made promises to demo the Sprints' results at the end of each Sprint. This may also have placed further pressure on the team to deliver even though the conditions were not optimal. The scope of **RQ3** was to examine what constrains the Scrum value-add; we did not seek to understand the root causes of each constraint. Future work could look at how project management decisions at the outset of projects cascade and break the Scrum value-add for software quality.
- **Case studies on constraints impacting software quality.** We assumed that the constraints identified in both cases impact only software quality. This is because our variable of interest is limited to software quality. This does not imply that these constraints do not have other implications. For example, inaccessibility to end users and team tensions could also impact team performance and the whole end-to-end development process efficiency.

8 Threats to Validity

Although some authors (e.g., Maxwell 1992; Wolcott 1990) have argued that "validity" is not suitable for qualitative work and suggested trustworthiness as an alternative concept, Miles et al. (2014) combine both views and suggest a set of techniques to establish the validity of qualitative studies. They cite objectivity/confirmability, reliability/dependability/audibility, internal validity, external validity, and utilization/application/action orientation as important quality criteria for qualitative work. Below, we discuss how we meet these requirements. We opted to simplify the labels proposed by Miles et al. (2014) for simplicity. We selected the most familiar label to a software engineering audience, e.g., reliability instead of reliability/dependability/audibility.

Objectivity To remain objective during the research process, we validated our findings using participants' feedback for Phase 1 conclusions. In Phase 2, we validated our interpretations of the observations data by ongoing informal conversations with different roles in the projects. The researcher engaged occasionally in informal conversations to align his interpretations with developers or other roles in the projects.

Miles et al. (2014) recommends including the study of "competing hypotheses" or "rival explanations" to ensure the objectivity of a qualitative study. To meet this requirement, we included potential explanations for phenomenon we observed and discussed them in the discussion section. Miles et al. (2014) also recommends "data retention." We made Phase 1 data publicly available including the analysis process and the output. During the recruitment process, we asked potential participants for their permission to make the interviews data available after anonymization. Unfortunately, Phase 2 data cannot be made available to the audience of this report. Both cases have made clauses in the legal agreement with us that prevent us from doing so.

Reliability Two techniques proposed by Miles et al. (2014) to establish reliability are "data quality" and "colleague review." We sent the interview transcripts to all Phase 1 participants for accuracy checks. Twenty-one participants advised no corrections, and four came up with

clarifications for some of their statements. Those corrections were made to the interviews transcripts. Some of the remaining participants either advised no need to check or did not respond. The initial coding was done by the first author, then the second author provided his comments and suggestions for additional codes.

Internal Validity Miles et al. (2014) proposes seeking "negative evidence" to establish internal validity and allowing the original participants to validate the accuracy of the conclusions. **RQ3** data is negative evidence that Scrum implementations do not always materialize the value put forward by Phase 1 participants. In Section 4.4, we juxtaposed the findings of **RQ2** and **RQ3** to propose an integrated model. We used participants' feedback to validate the accuracy of our conclusions.

Both Phase 2 cases have particularities, **Kolkata** project is an offshoring arrangement and **Phoenix** is an outsourcing one. Some of the constraints could be specific to these contractual set-ups, especially the inaccessibility of the business knowledge. However, similar issues could arise even within the same organization. Accessibility to the end-users or the participation of a competent product owner is not always guaranteed in agile teams, even in the same organization (Hoda et al. 2011). In addition, some of our Phase 1 participants also worked in similar arrangements. For example, participant 7 worked for a large American outsourcing provider with teams and clients across multiple countries. Participant 18 worked for a large Indian technology provider with clients based in the UK, USA, and Australia. Still, both reported enthusiastic accounts of their respective Scrum experiences. Our work shows that negative cases exist and the constraints their social environment exhibits can be shared with other cases (e.g., inaccessibility to the business knowledge) or unique to the company.

Our research design strategy also aimed to strengthen internal validity. We opted for combinations of negative and positive cases; this sampling strategy combined literal replication (selecting cases/informants with a similar outcome) with theoretical replication (selecting purposefully different cases, i.e., including two cases that did not manage to improve quality. We planned a sample that gave voice to key software development stakeholders, including management and product owners. Our sample had, though, its emphasis on software developers and QA personnel given the key importance of these roles for software quality. Although our cases (**Kolkata** and **Phoenix**) were selected conveniently, still, they are "typical" (Yin 2018) for **RQ3** purpose. To ensure the objectivity of this selection, we endeavored for two diverse cases. For example, both cases were from two different national cultures (Indian and American). In addition, both had distinct business objectives and organizational structures.

The COVID-19 Pandemic could have impacted both **Kolkata** and **Phoenix** cases. This variable was not included in our research question and, hence, was not a focus of our analysis. The pandemic may have influenced team morale, communication, and collaboration. Aware of this factor, the researcher asked on several occasions the developers whether the lockdown circumstances had contributed to the problems he observed. In an informal conversation with a senior developer from **Kolkata**, he stated: *"we communicate quite OK, but the issues we discussed before always existed. They are not new."* When a similar question was raised with **Phoenix** senior developer, he explained: *"obviously, few things have changed but mostly impacted our communication and coordination ... We use to set close to each other. The guys use to raise their heads and shout questions, no need for meetings or Skype calls. I have kids I look after at home. My wife goes to work ... I changed my working hours, I became less reliable and some of the guys in similar situation. I would say coordination of tasks has been affected and helping each other's. It's better when you*

can physically see the other person." This raised our confidence in **RQ3** findings. We used **RQ2** findings as a lens to examine both Phase 2 cases. This has led to findings relevant to further our understanding of what obstructs the Scrum value-add for achieving quality from materializing. The pandemic has relevance in these social settings; however, the constraints we identified for **RQ3** are not induced by lockdown circumstances. They rather stem from the cultural and organizational fabrics of both cases.

External Validity This is the quality of whether the findings can be "transferable" to other contexts (Miles et al. 2014). Miles et al. (2014) suggest discussing the contexts where the findings may apply and a theoretically diverse sample. In Section 6.1, we highlighted potential settings where these findings may be relevant. As discussed previously, we purposefully sought a diverse sample. Our Phase 1 participants were from 16 countries and worked for diverse types of businesses of various sizes.

Action Orientation As we argued throughout this report, Scrum has become a highly sought implementation of agile methods. This popularity is, partly, grounded in the motivation of companies to improve software quality by using Scrum. To influence future endeavors to implement Scrum to improve quality, we drew recommendations from our findings and aimed for actionable measures for practitioners to consider (see Section 6.1).

9 Conclusion

The Scrum value-add for software quality does not replace good software engineering practices, quality assurance, and control processes. Instead, it complements them and nurtures the social development environment to uplift the ability of Scrum teams to achieve software quality. However, this does not imply that the Scrum value-add is inessential; our data show significant appraisal for their relevance in the quest to achieve software quality. The organizational and cultural fabric of organizations influences Scrum implementations and the potential for the Scrum value-add to materialize. Cultural particularities and leadership styles have a far-reaching impact on software quality. For example, we reported that the Scrum value-add is impeded when team members avoid speaking up, lack openness, and blindly accept the leader's decisions, or fear negative consequences. Consequently, Scrum teams miss the opportunities to raise and work on their obstacles to improve their ability to improve the quality of their work.

Our work demonstrates that Scrum implementations do not "work in vacuum." They have a social underpinning that complements and strengthens their effectiveness and encourages a social climate in favor of achieving software quality. Organizations seeking to enhance their software quality by implementing Scrum should combine software engineering and quality assurance practices with a focus on a social climate that promotes the Scrum value-add for achieving software quality.

Appendix 1

All information that may compromise the anonymity of Phase 1 participants or both companies of Phase 2 has been masked. All screen shots have been reviewed and approved by the participants, **Kolkata** and **Phoenix** points of contact for inclusion in this paper.

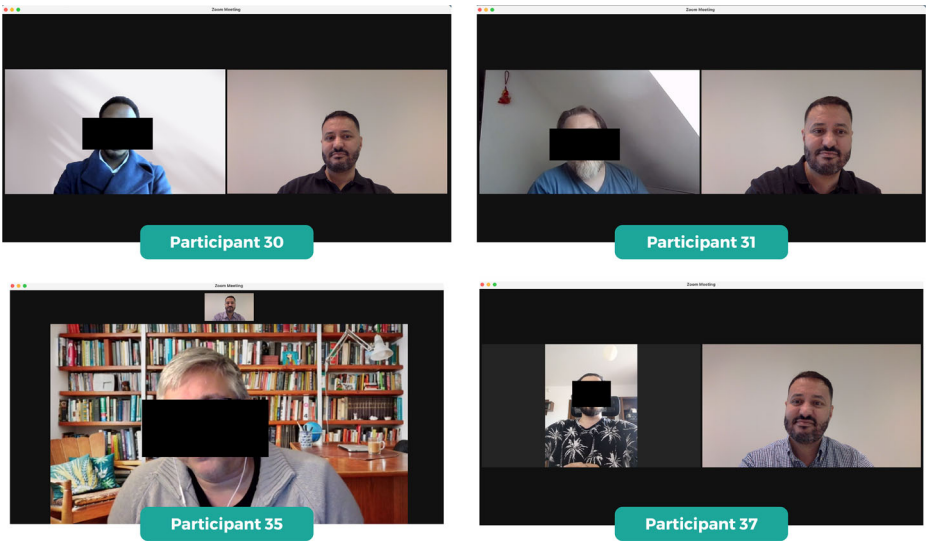

Fig. 5 Evidence from fieldwork - Phase 1 interviews with participant 30, 31, 35 & 37

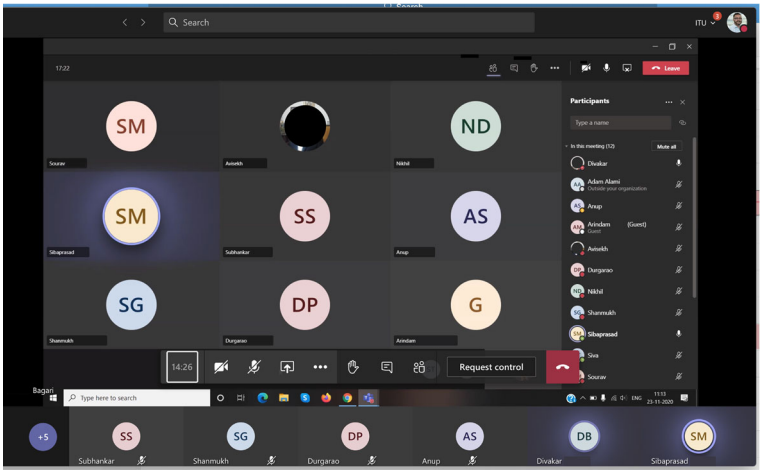

Fig. 6 Evidence from fieldwork - Phase 2 observations, **Kolkata** Standup organized on 20/11/2020

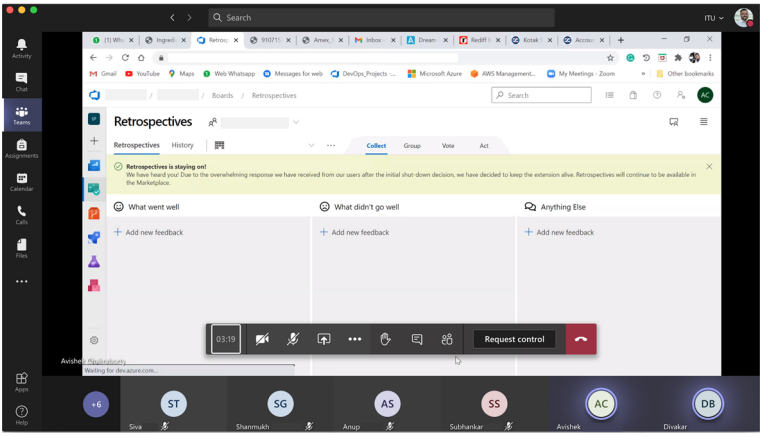

Fig. 7 Evidence from fieldwork - Phase 2 observations, **Kolkata** retrospective organized on 12/11/2020

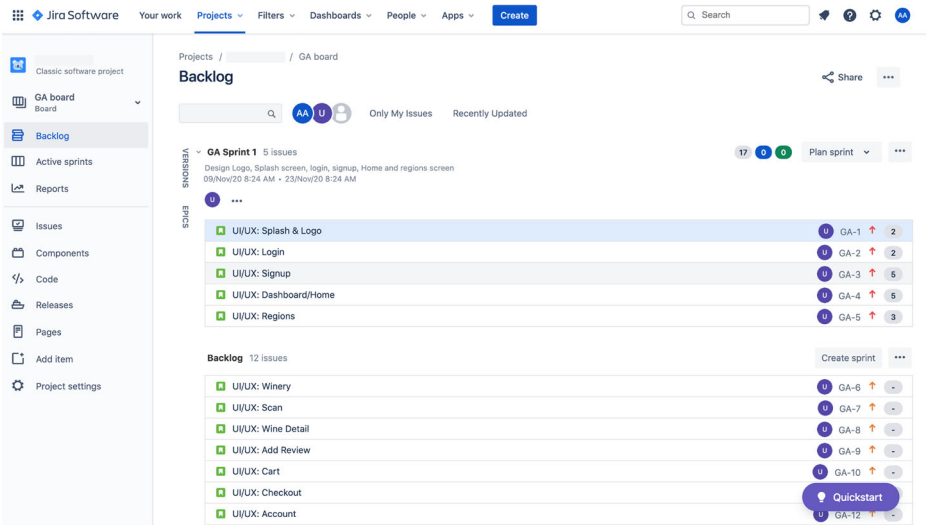

Fig. 8 Evidence from fieldwork - Phase 2 observations, **Phoenix** Sprint 1 product Backlog

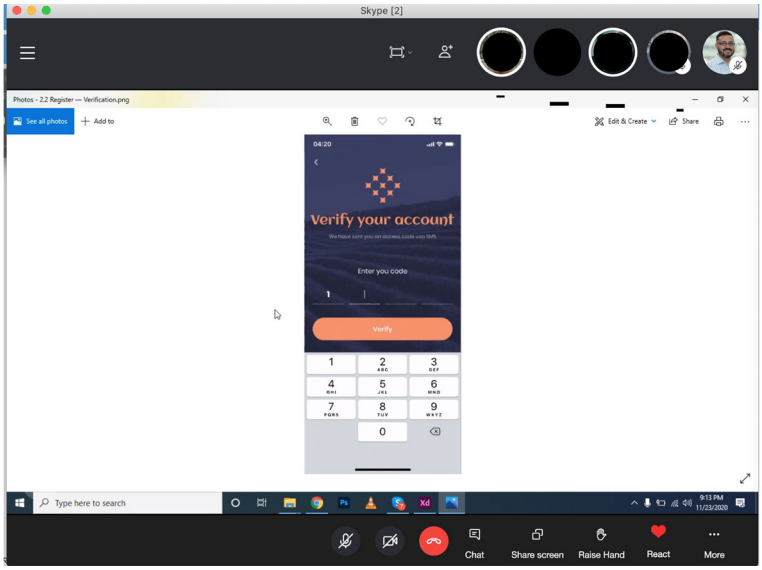

Fig. 9 Evidence from fieldwork - Phase 2 observations, **Phoenix** internal product demo organized on 23/11/2020

Acknowledgments We would like to thank the interviewees for their time and inputs. We also would like to thank our cases, **Kolkata** and **Phoenix**.

References

Abbas N, Gravell AM, Wills GB (2010a) The impact of organization, project and governance variables on software quality and project success. In: 2010 Agile conference. IEEE, pp 77–86

Abbas N, Gravell AM, Wills GB (2010b) Using factor analysis to generate clusters of agile practices (a guide for agile process improvement). In: 2010 Agile conference. IEEE, pp 11–20

Alami A, Paasivaara M (2021) How do agile practitioners interpret and foster "technical excellence"? In: Evaluation and assessment in software engineering. ACM, pp 10–19

Alami A, Wąsowski A (2019) Affiliated participation in open source communities. In: 2019 ACM/IEEE International symposium on empirical software engineering and measurement (ESEM). IEEE, pp 1–11

Alami A, Cohn ML, Wąsowski A (2020) How do foss communities decide to accept pull requests? In: Proceedings of the evaluation and assessment in software engineering. ACM, pp 220–229

Alami A, Krancher O, Paasivaara M (2022) The journey to technical excellence in agile software development. Information and Software Technology 106959

Ambler SW (2008) http://www.ambysoft.com/surveys/agileFebruary2008.html

Arcos-Medina G, Mauricio D (2019) Aspects of software quality applied to the process of agile software development: a systematic literature review. Int J Syst Assurance Eng Manag 10(5):867–897

Bag PK, Pepito N (2011) Double-edged transparency in teams. J Public Econ 95(7-8):531–542

Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R et al (2001) Manifesto for agile software development. Agile Manifesto

Benbasat I, Goldstein DK, Mead M (1987) The case research strategy in studies of information systems. MIS Q 369–386

Bjarnason E, Gislason Bern B, Svedberg L (2022) Inter-team communication in large-scale co-located software engineering: a case study. Empir Softw Eng 27(2):1–43

Bovens M (2007) Analysing and assessing accountability: a conceptual framework 1. Eur Law J 13(4):447–468

Burga R, Spraakman C, Balestreri C, Rezania D (2021) Examining the transition to agile practices with information technology projects: agile teams and their experience of accountability. Int J Project Manag
Çaglayan B, Bener AB (2016) Effect of developer collaboration activity on software quality in two large scale projects. J Syst Softw 118:288–296
Chong J (2005) Social behaviors on xp and non-xp teams: a comparative study. In: Agile development conference (ADC'05). IEEE, pp 39–48
Chuang S-W, Luor T, Lu H-P (2014) Assessment of institutions, scholars, and contributions on agile software development (2001–2012). J Syst Softw 93:84–101
Cohn M (2004) User stories applied: for agile software development. Addison-Wesley Professional
Creswell JW, Miller DL (2000) Determining validity in qualitative inquiry. Theory Pract 39(3):124–130
Creswell JW, Poth CN (2016) Qualitative inquiry and research design: choosing among five approaches. Sage Publications
Deemer P, Benefield G, Larman C, Vodde B (2012) A lightweight guide to the theory and practice of scrum. Ver 2:2012
digital.ai (2021) 15th state of agile report: agile leads the way through the pandemic and digital transformation. https://digital.ai/catalyst-blog/15th-state-of-agile-report-agile-leads-the-way-through-the- pandemic-and-digital
Dingsøyr T, Nerur S, Balijepally V, Moe NB (2012) A decade of agile methodologies: towards explaining agile software development. J Syst Softw 6(85):1213–1221
Duhigg C (2016) What Google learned from its quest to build the perfect team. New York Times Mag 26(2016):2016
Edmondson A (1999a) Psychological safety and learning behavior in work teams. Adm Sci Q 44(2):350–383
Edmondson A (1999b) A safe harbor: social psychological conditions enabling boundary spanning in work teams. Elsevier Science/JAI Press
Edmondson A (2018) The fearless organization: creating psychological safety in the workplace for learning, innovation, and growth. Wiley
Eisenhardt KM (1989) Agency theory: an assessment and review. Acad Manag Rev 14(1):57–74
Eloranta V-P, Koskimies K, Mikkonen T (2016) Exploring scrumbut—an empirical study of scrum anti-patterns. Inf Softw Technol 74:194–203
Emigh RJ (1997) The power of negative thinking: the use of negative case methodology in the development of sociological theory. Theory Soc 26(5):649–684
English M, Buckley J, Collins J (2016) Investigating software modularity using class and module level metrics. In: Software quality assurance. Elsevier, pp 177–200
Gibson CB, Gibbs JL (2006) Unpacking the concept of virtuality: the effects of geographic dispersion, electronic dependence, dynamic structure, and national diversity on team innovation. Adm Sci Q 51(3):451–495
Green P (2011) Measuring the impact of scrum on product development at adobe systems. In: 2011 44th Hawaii international conference on system sciences. IEEE, pp 1–10
Hanson A (2017) Negative case analysis. The international encyclopedia of communication research methods, pp 1–2
Hanssen GK, Fægri TE (2006) Agile customer engagement: a longitudinal qualitative case study. In: Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering, pp 164–173
Hassani-Alaoui S, Cameron A-F, Giannelia T (2020) "We use scrum, but...": agile modifications and project success. In: Proceedings of the 53rd Hawaii international conference on system sciences
Hennel P, Rosenkranz C (2021) Investigating the "socio" in socio-technical development: the case for psychological safety in agile information systems development. Proj Manag J 52(1):11–30
Hoda R, Noble J (2017) Becoming agile: a grounded theory of agile transitions in practice. In: 2017 IEEE/ACM 39th international conference on software engineering (ICSE). IEEE, pp 141–151
Hoda R, Noble J, Marshall S (2011) The impact of inadequate customer collaboration on self-organizing agile teams. Inf Softw Technol 53(5):521–534
Hofstede G (1984) Culture's consequences: international differences in work-related values, vol 5. Sage
Hofstede G (2001) Culture's consequences: comparing values, behaviors, institutions and organizations across nations. Sage Publications
Ilieva S, Ivanov P, Stefanova E (2004) Analyses of an agile methodology implementation. In: Proceedings. 30th Euromicro conference, 2004. IEEE, pp 326–333
ISO/IEC (2011) Iso/iec 25010:2011(en) systems and software engineering—systems and software quality requirements and evaluation (square)—system and software quality models. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

Issac G, Rajendran C, Anantharaman R (2003) Determinants of software quality: customer's perspective. Total Qual Manag Bus Excell 14(9):1053–1070
Jain P, Sharma A, Ahuja L (2018) The impact of agile software development process on the quality of software product. In: 2018 7th International conference on reliability, Infocom technologies and optimization (trends and future directions)(ICRITO). IEEE, pp 812–815
Jalote P, Agrawal N (2005) Using defect analysis feedback for improving quality and productivity in iterative software development. In: 2005 International conference on information and communication technology. IEEE, pp 703–713
Khadjavi M, Lange A, Nicklisch A (2014) The social value of transparency and accountability: experimental evidence from asymmetric public good games. ZBW-deutsche Zentralbibliothek für..., Kiel und Hamburg
Kitchenham B, Pfleeger SL (1996) Software quality: the elusive target [special issues section]. IEEE Softw 13(1):12–21
Kitchenham B, Brereton OP, Budgen D, Turner M, Bailey J, Linkman S (2009) Systematic literature reviews in software engineering–a systematic literature review. Inf Softw Technol 51(1):7–15
Krancher O (2020) Agile software development practices and success in outsourced projects: the moderating role of requirements risk. In: International conference on agile software development. Springer, pp 56–72
Krancher O, Luther P, Jost M (2018) Key affordances of platform-as-a-service: self-organization and continuous feedback. J Manag Inf Syst 35(3):776–812
Krasner H (2018) Research report. https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report/
Kraus WA, Kraus WA (1980) Collaboration in organizations: alternatives to hierarchy. Human Sciences Press, New York
Layman L, Williams L, Cunningham L (2004) Exploring extreme programming in context: an industrial case study. In: Agile development conference. IEEE, pp 32–41
Li J, Moe NB, Dybå T (2010) Transition from a plan-driven process to scrum: a longitudinal case study on software quality. In: Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement, pp 1–10
Lincoln YS, Guba EG (1985) Establishing trustworthiness. Natural Inq 289(331):289–327
Mahoney J, Goertz G (2004) The possibility principle: choosing negative cases in comparative research. Am Polit Sci Rev 98(4):653–669
Maximilien EM, Williams L (2003) Assessing test-driven development at ibm. In: 25th International conference on software engineering, 2003. Proceedings. IEEE, pp 564–569
Maxwell J (1992) Understanding and validity in qualitative research. Harv Educ Rev 62(3):279–301
Maxwell J (2012) Qualitative research design: an interactive approach. Sage Publications
May DR, Gilson RL, Harter LM (2004) The psychological conditions of meaningfulness, safety and availability and the engagement of the human spirit at work. J Occup Organ Psychol 77(1):11–37
McHugh O, Conboy K, Lang M (2011) Agile practices: the impact on trust in software project teams. IEEE Softw 29(3):71–76
Mero-Jaffe I (2011) 'Is that what I said?' Interview transcript approval by participants: an aspect of ethics in qualitative research. Int J Qual Methods 10(3):231–247
Messner M (2009) The limits of accountability. Acc Organ Soc 34(8):918–938
Miles MB, Huberman AM, Saldana J et al (2014) Qualitative data analysis: a methods sourcebook. Sage, Thousand Oaks
Müller R (2010) Project governance. Routledge
Oliver G (2011) Organisational culture for information managers. Elsevier
Patton MQ (1999) Enhancing the quality and credibility of qualitative analysis. Health Serv Res 34(5 Pt 2):1189
Patton MQ (2014) Qualitative research & evaluation methods: integrating theory and practice. Sage Publications
Prechelt L, Schmeisky H, Zieris F (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, pp 1017–1027
Roberts J (1991) The possibilities of accountability. Account Organ Soc 16(4):355–368
Rodríguez P, Markkula J, Oivo M, Turula K (2012) Survey on agile and lean usage in Finnish software industry. In: Proceedings of the 2012 ACM-IEEE international symposium on empirical software engineering and measurement. IEEE, pp 139–148
Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14(2):131–164
Saldaña J (2021) The coding manual for qualitative researchers. Sage

Schein E, Bennis WG (1965) Personal and organizational change through group methods: the laboratory approach, Wiley, New York
Schwaber K, Sutherland J (2017) The definitive guide to scrum: the rules of the game. Scrum.org
Schwaber K, Sutherland J (2020) The scrum guide. Scrum Alliance 21(19):1
Sharp H, Dittrich Y, De Souza CR (2016) The role of ethnographic studies in empirical software engineering. IEEE Trans Softw Eng 42(8):786–804
Sutherland J, Sutherland J (2014) Scrum: the art of doing twice the work in half the time. Currency
Sutherland J, Schoonheim G, Rijk M (2009) Fully distributed scrum: replicating local productivity and quality with offshore teams. In: 2009 42nd Hawaii international conference on system sciences. IEEE, pp 1–8
Tarhan A, Yilmaz SG (2014) Systematic analyses and comparison of development performance and product quality of incremental process and agile process. Inf Softw Technol 56(5):477–494
Teasley S, Covi L, Krishnan MS, Olson JS (2000) How does radical collocation help a team succeed? In: Proceedings of the 2000 ACM conference on computer supported cooperative work, pp 339–346
Tucker AL, Nembhard IM, Edmondson AC (2007) Implementing new practices: an empirical study of organizational learning in hospital intensive care units. Manag Sci 53(6):894–907
ul Musawir A, Abd-Karim SB, Mohd-Danuri MS (2020) Project governance and its role in enabling organizational strategy implementation: a systematic literature review. Int J Proj Manag 38(1):1–16
Vijayasarathy L, Turk D (2008) Agile software development: a survey of early adopters. J Inf Technol Manag 19(2):1–8
Williams L, Brown G, Meltzer A, Nagappan N (2011) Scrum+ engineering practices: experiences of three microsoft teams. In: 2011 International symposium on empirical software engineering and measurement. IEEE, pp 463–471
Wolcott HF (1990) On seeking-and rejecting-validity in qualitative research. Qualitative inquiry in education: the continuing debate, pp 121–152
Yarnold PR, Mueser KT, Lyons JS (1988) Type a behavior, accountability, and work rate in small groups. J Res Pers 22(3):353–360
Yin RK (2018) Case study research and applications. Sage