Adapt and Overcome - How Agile Practitioners Adapt to Issues that Impede the Delivery of Value: An Interview Study

Jan-Niklas Meckenstock[1(✉)] and Victoria Wallmichrath[2]

[1] Chair for Industrial Information Systems, University of Bamberg, Bamberg, Germany
jan-niklas.meckenstock@uni-bamberg.de
[2] Babtec Informationssysteme GmbH, Wuppertal, Germany

Abstract. Continuously delivering valuable software is a core principle of agile software development (ASD). In practice, value delivery is often impeded by several key issues, including low customer involvement, volatile requirements, technical debt, delivery pressure, excessive rework, or meeting overhead, which affect the product or the process. Despite their negative influence, the different consequences for the delivered value remain to be better understood. In addition, a collection of measures to help practitioners adapt to these issues is missing, as previous work only offers limited guidance on how to mitigate them. To address this situation, we conducted 19 semi-structured expert interviews to identify the consequences of key issues for value delivery and empirically derive measures to adapt to these issues. We find 34 value-reducing consequences, which primarily affect product quality and capabilities, delivery timeliness, and process efficiency. We also develop a collection of 48 measures to address the issues, including procedural changes, process artifacts and roles, technical means, and different ways to approach customers. With our work, we provide practitioners with actionable measures to adapt to issues encountered in daily practice and avoid their value-reducing consequences, thereby facilitating continuous value delivery. For research, we extend knowledge on the dark side of ASD by illustrating how key issues affect the delivered value, which was less regarded in related studies. In addition, we encourage future investigations into how micro-tailoring fosters sustained value delivery, along with examinations of resilience engineering in the context of ASD to improve the performance of software development processes.

Keywords: Agile software development · Agile business value · Issues impeding value delivery · Countermeasures to sustain value delivery · Interview study

1 Introduction

Agile software development (ASD) methodologies such as Scrum, Extreme Programming (XP), or Kanban have transformed the software development (SD) industry, receiving considerable attention from various research communities [3, 12]. At its core, ASD was conceived to "*satisfy the customer through early and continuous delivery of valuable*

S. Peter et al. (Eds.): XP 2025, LNBIP 545, pp. 245–264, 2025.
https://doi.org/10.1007/978-3-031-94544-1_17

software" [4], as proclaimed in the Agile Manifesto. Research has revealed significant benefits that form the value of ASD [25], particularly with regard to the provided software products. Among other benefits, these include higher product quality with fewer defects [2, 16], reduced time to market [26, 29], or better meeting of requirements [2, 26]. Together, these benefits contribute to the steadily increasing adoption of ASD methodologies, as frequently reported in the annual State of Agile Reports [11].

In practice, "*continuous delivery of valuable software is not an easy task*," [18] as various issues [24, 28] or barriers [1] can impede value delivery. In this regard, a literature study by Meckenstock [24] systematizes issues that occur frequently in ASD. *Lacking customer involvement, technical debt, volatile requirements, delivery pressure, required rework,* and *meeting overhead* appear especially problematic [24], inter alia, due to their potential effects on the timeliness of deliverables [17], product quality [13], or the correctness of requirements [19]. In analogy, a study by Alahyari et al. [1] on value in ASD finds several barriers that can impede value delivery. As a commonality, however, the studies do not thematize the consequences for value delivery if these issues or barriers are not overcome. Similarly, Petersen and Wohlin [28] investigated benefits and issues in ASD, but did not assess the specific consequences of the identified issues for value delivery. Besides lacking insights into the consequences for the delivery of value, prior work also did not provide measures to adapt to these issues. Identifying the consequences of key issues in ASD and adapting to them appears important, though, to ensure the continuous delivery of value and improve the application of ASD.

To address the lack of insights into the consequences of six key issues for value delivery and provide a collection of measures that can help practitioners uncover better ways to deliver valuable software products, we conducted an interview study with 19 ASD experts. Our study investigates the following research questions (RQ):

- *RQ1: What are the consequences of key issues in ASD methodologies for the sustained delivery of valuable software products?*
- *RQ2: Which measures do practitioners apply to adapt to critical issues and avoid their negative consequences that affect a sustained value delivery?*

The paper is structured as follows: in Sect. 2, we describe the theoretical background of the study and related work. Section 3 contains the methodology behind our work. Section 4 presents the results of the study and illustrates the consequences of the six issues and measures to adapt to them. Section 5 discusses the key findings, suggests paths for future research, and addresses limitations. Section 6 concludes the paper.

2 Theoretical Background and Related Work

Delivering value to the customer is deeply rooted in the principles of the Agile Manifesto [4], making it a core objective of ASD methodologies. The Scrum guide emphasizes this value delivery by continuously providing increments towards a product goal [34], i.e., a software product, with ASD methodologies generally "*contributing to perceived customer value*" [9]. The term *value* is often perceived differently, but implies a "*benefit, whether tangible or intangible, economic or social, monetary or utilitarian, […] aesthetic or ethical*" [5] that can be derived from such a software product. While IS research,

for instance, views value from a more economic-financial stance [33], research on the value of ASD adopts a wider perspective, emphasizing that "*value is not only dollars*" [29]. In this vein, [25] find that the value of delivered products materializes in the following ways: better meeting of customer demands through frequent feedback, higher software quality, reduced time to market, better responsiveness to changing requirements, and increased productivity, among other non-product-related benefits, i.e., developer satisfaction [29] or improved communication [2]. In analogy, [1] suggest that ASD value is mostly perceived to manifest in product functionality, maintainability, quality, and time to availability, with financial aspects being less frequently mentioned. While ASD can also imply other beneficial effects [1, 25, 26, 29], our study focuses on product-delivery-related benefits, given the promise of ASD to continuously provide valuable software [4, 34]. In line with prior research, we consider the core value proposition of ASD to imply an *early, frequent, and sustained delivery of a high-quality software product that can be evaluated with the customer in short iterations, leading to a better meeting of requirements through higher responsiveness to change.*

In practical settings, several issues impede the delivery of valuable products and affect ASD's core value proposition. Six issues seem especially critical [24], as they affect the product, reinforce other issues, or imply negative effects on the process. These include *lacking customer involvement, technical debt, volatile requirements, delivery pressure, required rework,* and *meeting overhead*. Similarly, potential barriers that impede value delivery found by Alahyari et al. [1] align with the issues discussed in Meckenstock [24], i.e., vague requirements, late scope changes, lack of access to customers, or tight deadlines [1]. Below, we describe these critical concerns for value delivery:

Lacking customer involvement is problematic, as ASD teams rely on the customer to clarify and prioritize requirements and seek feedback on recent increments [19] to align the product with the requirements. *Volatile requirements* lead to inadequate estimates [30] that often cause delays [17]. In addition, *delivery pressure* results in shortcuts in development that degrade product quality and create technical debt [23]. *Technical debt* implies deficiencies in the code resulting from time pressure [23], which hampers product quality and maintainability in the long run [13]. *Required rework* of increments due to missed requirements and technical debt [24] may also be problematic, as it increases project costs [37]. Finally, despite their importance in steering the development process, *meeting overhead* disrupts the workflow and reduces developer productivity [39].

These six issues all imply certain negative effects, but how they specifically affect the delivery of valuable software is less clear. While [24] illustrates which issues occur most frequently and how they are related, the specific consequences for the delivered value are less regarded. Similarly, [1] only list barriers toward achieving value, yet the consequences, if these barriers are not overcome, remain unaddressed. Together, previous studies only describe which issues generally occur and which barriers exist in ASD, with the specific consequences for value delivery being less understood.

In addition, previous work does not provide a comprehensive collection of measures to adapt to these issues. Adaptation is a key principle of ASD, though, as suggested in the key managing rule of XP to "fix XP when it breaks" [38]. Therefore, inspecting and adapting the process to fix aspects that are not working well is embedded in virtually all ASD methodologies, i.e., XP or Scrum [3, 38]. Still, [1] only describe practices that

should generally be in place for value creation, while measures to overcome the identified barriers are missing. This also holds true for [24], which generally does not thematize how to adapt to issues to ensure value delivery. With regard to adaptation, research on ASD method tailoring [8, 36], i.e., "*customizing the agile method to meet the context and circumstances of use*" [36] also primarily focuses on tailoring motives and criteria [8], the effects of tailoring on success [36], and the main approaches of contingency-based method selection or method engineering [8]. How to customize ASD with measures or micro-adaptations [16] to sustain value delivery is less investigated, though. While some suggestions are scattered in the literature, i.e., how to address low customer involvement [19] or technical debt [23], a comprehensive set of measures to help ASD teams adapt to key issues thus remains to be defined.

To address these shortcomings, this study sheds light on the consequences of the six key issues derived from [24] for the core proposition of ASD to deliver valuable software products. In so doing, we aim to extend findings on the value [1, 25] and the dark sides of ASD [24] by clarifying the specific impact of the examined issues on the delivered value. Our study also intends to provide an empirically-derived collection of measures that practitioners can apply to adapt to these issues, as a consolidated set of practical measures is lacking. Thereby, we also contribute insights into how ASD practitioners adapt to critical issues, which was missing in [1, 24]. Together, this study should equip ASD practitioners with actionable measures to help them find better ways to deliver valuable software products when facing these issues in daily practice.

3 Methodology and Study Design

Study Design. To answer our RQs, we conducted semi-structured expert interviews [6], relying on the guidelines by Myers and Newman [27] to enable a rigorous data collection. Doing so appears suitable, as we investigate a variety of practical problems [35] that ASD practitioners face in daily development practice. An interview protocol with three sections guided the interviews to ensure that all experts were asked a consistent set of questions. The interview protocol is featured in the online appendix[1]. After the first three interviews, we refined the protocol to address observed gaps in the questions and achieve better alignment with the research objectives. In each interview, we first presented the goal of the study and asked the experts about their experience with ASD, the methodologies they apply, and their perception of the value of ASD. In the second part, we asked about the negative consequences of each investigated issue derived from [24] for the delivery of valuable software. In this section, we also inquired about how the experts specifically adapted to these issues to continuously deliver valuable software. In the wrap-up section, we asked which issue the experts deemed to be the most problematic. We also inquired if they wanted to add an issue or a consequence that was not mentioned before, however, most experts considered the six investigated issues as the main sources of negative consequences for a sustained value delivery.

Data Collection. The interview study was conducted between September and December 2024. We performed 19 interviews with experts from 14 organizations to gain a

[1] The online appendix can be found at: https://doi.org/10.6084/m9.figshare.28190441.

Table 1. Overview of participating ASD experts

ID	Role	Country	Industry	Agile Experience	No. of Employees	Applied Agile Methodologies
E1	Agile Coach	USA	Banking	13 years	50.000	SAFe, Scrum
E2	Developer	Switzerland	Groceries and Retail	4 years	2.200	SAFe, Scrum
E3	Agile Coach	Australia	Insurance	9 years	90.000	Scrum
E4	Scrum Master	Germany	ICT Services	4 years	1.000	Scrum
E5	Business Analyst	Germany	IT Consulting	4 years	300.000	Scrum
E6	Developer	Portugal	IT Consulting	3 years	370.000	Scrum
E7	Scrum Master	Germany	ICT Services	7 years	1.000	Scrum
E8	Product Owner	Germany	ICT Services	6 years	1.000	Scrum
E9	Developer	Germany	Banking	5 years	1.100	Scrum
E10	Head of Software	Germany	ICT Services	15 years	250	Scrum
E11	Product Owner	Germany	HR Services	5 years	50	Scrum
E12	Developer	Switzerland	ICT Services	6 years	180.000	Scrum, Kanban
E13	Head of Technology	Germany	Marketing	11 years	200	Scrum, Kanban
E14	Agile Coach	Germany	ICT Services	12 years	450	Scrum, Kanban
E15	Agile Coach	Germany	Healthcare	13 years	300.000	Scrum, Kanban, Lean
E16	Scrum Master	Germany	ICT Services	8 years	1.000	Scrum
E17	Developer	Germany	ICT Services	8 years	250	Scrum
E18	Head of Software	Spain	ICT Services	17 years	20	Scrum
E19	Head of Software	Germany	ICT Services	15 years	250	Scrum

broad perspective on the topic with a "*variety of voices*" [27]. Following the definition of *expert* in [6], we considered an *expert* in ASD as someone who "*has technical, process, and interpretive knowledge [in the field of agile development], by virtue of the fact that the expert acts [as a developer, coach, Scrum Master or Product Owner, i.e., any role associated with the application of ASD]*", (cf. [6], p. 54). Table 1 contains an overview of all experts [E1-E19]. This selection ensured that diverse perspectives on issues, their consequences, and applicable adaptation measures could be obtained. The experts were recruited using a combination of purposive and convenience sampling [20] and approached via LinkedIn or referred to the authors after an interview. All interviews were conducted in Microsoft Teams and recorded after the experts had given their consent, with the first author serving as the interviewer in all cases. On average, the interviews lasted 43 min, ranging from 32 to 58 min. Most interviews were conducted in German, while five were in English. The recordings were subsequently transcribed, with all German interviews being translated into English for the analysis step. The first author transcribed and translated the interviews, which the second author later validated, while potential disagreements were resolved in joint discussions.

Data Analysis. The collected data was analyzed in a three-step cycle with inductive and deductive characteristics. We followed the recommendations by Saldaña [32] and Kuckartz [21] to identify relationships between the six issues and their consequences for value delivery as well as measures to adapt to these issues. The qualitative analysis was performed in MAXQDA, with the two researchers separately coding the data to avoid bias and ensure reliability, while disagreements were again resolved in joint discussions. In the first step, we identified negative consequences for the delivered value based on 1108 statements from the interviews. The consequences were assigned to the specific issue of the six investigated in this study that caused them. In the second step, we distilled measures the experts applied to adapt to these issues. In the third step, we revisited the initial results and refined them for the results presentation. Overall, we found 34 value-reducing consequences to result from the six issues. We also merged related consequences into 9 consequence types to allow further abstraction, see Table 2. Furthermore, we derived 48 measures that the experts used to adapt to the six issues, with related measures grouped into 7 measure types, see Table 3. The online appendix features a more detailed account of the analysis and provides the code book behind our work. We also sent out a follow-up report to five interested experts to reduce misinterpretations, receiving three responses that clarified statements we deemed as difficult to understand. One expert supplied an additional list of points to further expand on statements that lacked information, thereby enriching the existing data. The preliminary results were also presented to three experts to allow for further validation of the study.

4 Findings

In this section, we first illustrate the value-reducing consequences of the six examined issues. All consequences are listed in Table 2. Subsequently, the measures the experts applied to adapt to the key issues are described. Table 3 contains all identified measures.

4.1 Negative Consequences of Key Issues for Value Delivery

Delivery Pressure had a particularly negative effect on software quality, as developers "*cut corners and focus on the wrong thing, which is deliver on time rather than deliver a good outcome*" [E3]. Under pressure, also technical deficiencies in the code accumulated, since having to provide a continuous "*feature firework*" [E8] often resulted in the negligence of testing and other quality measures [E6,E7,E8,E11]. Besides poor quality, "*pressure creates anxiety*" [E1] and endangers psychological safety [E7,E10,E12,E16], which caused developers to experience stress and health issues [E2,E7,E8,E16]. As a related consequence, the delivered scope shrunk, given the team's capacity was reduced due to pressure-related cases of illness, causing them not to deliver to the full extent [E8,E16]. In addition, when pressurized, some teams focused on providing fewer features to avoid sacrificing quality, again at the expense of the scope [E5,E8,E12]. Most developers saw delivery pressure as the most problematic issue, i.e., [E2,E6,E12,E17].

Lacking Customer Involvement had negative consequences for the specification of requirements, the delivered functionality, and the timeliness of deliveries. Delivered products were often misaligned with requirements, as teams built features that are "*not in line with what was actually desired*" [E2]. Low customer engagement also impeded teams from receiving feedback [E2,E8,E17,E19] to assess the delivered increments. Besides deviations from actual requirements, obsolete functionality "*that nobody uses and then ends up in the bin*" [E16] resulted, further suggesting a waste of resources [E6,E9,E13,E16]. Several experts also cited incomplete requirements as a source of this issue, i.e., [E2,E4,E6,E9,E16], causing teams to operate in "*the fog of uncertainty*" [E5,E6] due to a lack of customer feedback. Finally, low customer involvement slowed development, causing delays when "*decisions are pending that would define subsequent steps*" [E17]. Especially Product Owners and Scrum Masters [E3,E4,E7,E8,E16] deemed low customer involvement as the most critical issue, since "*Product Owners cannot challenge their requirements and have to pull something out of thin air*" [E4].

Meeting Overhead and Interruptions affected the developers' concentration and focus, especially during coding work, as "*getting back into what you had been working on before is often difficult and takes time*" [E12]. Other experts [E8,E9,E16,E18] also reported frustration when spending much of their day in meetings, as it took away time for the development of functionality and implied an interruption of the workflow [E2, E4,E6,E12,E17]. Overall, many experts perceived meetings as a loss of time [E3,E8,E9, E12,E15,E17], but particularly those with large numbers of participants, lacking goals and minimal speaking time for some of the members. For the most critical consequence, developers reported reduced productivity, as "*not only the duration of the meeting itself but also the time before and afterward impedes us from making actual progress*" [E17].

Required Rework was identified by [24] to be an issue causing further problems, for instance, cost increases. However, most experts saw rework as a value-enhancing aspect rather than being problematic. [E3] described rework "*as the only way to get value*", but only if it is "*a meaningful optimization or extension of extant work*" [E10]. [E19] added that "*rework improves the product, but the extra time required may come at a cost*", in line with [E2,E14]. The experts generally highlighted that it is essential to differentiate the kind of rework that occurs, i.e., whether amendments to existing features deliver a value add or merely imply extra work that "*rather represents gold-plating*" [E10]. As

will be described in Sect. 4.2, using a good / bad / ugly approach when assessing rework was suggested by the experts [E7,E10,E17] to ensure that only value-adding rework is

Table 2. Observed consequences of examined issues for value delivery

Key Issue	Consequence Type	Consequence	Expert IDs
Delivery Pressure	Additional Efforts	Additional Work Required	E2, E12, E14, E16
	Emotional Issues	Stress and Health Issues	E1, E2, E5, E6, E7, E8, E9, E11, E12, E14, E16, E17, E18
	Product Quality	Reduced Code Quality	E1, E2, E3, E4, E6, E7, E8, E11, E12, E13, E14, E15, E17, E19
	Product Capability	Reduced Delivery Scope	E5, E8, E12, E14, E16
Lacking Customer Involvement	Emotional Issues	Demotivation of Developers	E4, E6, E9, E17
	Product Capability	Obsolete Functionality	E6, E9, E13, E16
		Product-Requirement-Misalignment	E2, E3, E4, E5, E8, E9, E11, E16, E17, E19
	Requirement Issues	Unspecific / Incomplete Requirements	E2, E3, E4, E5, E6, E8, E9, E12, E15, E16, E17
	Time-to-Market	Delay of new Features	E2, E4, E6, E9, E15, E17
	Waste Issues	Waste of Resources	E6, E9, E13, E16
Meeting Overhead and Interruptions	Emotional Issues	Developer Frustration	E8, E9, E12, E16, E18
	Process Efficiency	Loss of Focus	E4, E6, E8, E9, E11, E12, E13, E17, E19
		Reduced Productivity	E2, E6, E8, E12, E19
		Workflow Interruption	E2, E4, E6, E12, E17
	Waste Issues	Loss of Time	E1, E3, E7, E8, E9, E12, E15, E17
Required Rework	Additional Efforts	Additional Time Required	E2, E9, E14
Technical Debt	Additional Efforts	Additional Cost	E1, E2, E6, E9, E12, E17, E19
		Additional Rework	E1, E4, E6, E10, E12, E14, E17

(*continued*)

Table 2. (*continued*)

Key Issue	Consequence Type	Consequence	Expert IDs
	Customer Satisfaction	Deteriorating Customer Relationship	E2, E14, E16
	Process Efficiency	Reduced Velocity	E7, E9, E16, E17, E18
	Product Quality	Difficult Code Understanding	E2, E8, E12, E16, E17
		Difficult Maintainability	E9, E12, E16, E17
		Reduced Software Quality	E2, E6, E12, E14, E15, E16, E19
		Security Issues	E5, E9, E10, E15, E16
	Time-to-Market	Delay of new Features	E4, E6, E7, E9, E14, E16, E17
	Waste Issues	Loss of Time	E2, E4, E7, E8, E11, E14, E19
Volatile and Unclear Requirements	Additional Efforts	Additional Cost	E2, E4, E5, E7
	Emotional Issues	Frustration of Developers	E6, E8, E14, E15, E16, E17
	Product Capability	Incoherent Functionality	E7, E12, E16, E17, E19
		Reduced Delivery Scope	E2, E11, E16
	Product Quality	Technical Error	E2, E4, E11, E12, E16, E17
	Requirement Issues	Feature Creep	E5, E10, E16
	Time-to-Market	Delay of new Features	E6, E9, E10, E11, E16, E19
	Waste Issues	Wasted Time and Effort on Obsolete Functionality	E7, E8, E9, E14

performed, which follows [15]'s good-bad-ugly-taxonomy for iterative rework. Hence, rework is essential for value but requires adequate consideration.

Technical Debt was especially problematic for quality-related aspects. The experts reported that software quality declined, i.e., [E2,E6,E12,E15,E19], with cross-cutting concerns such as security issues, scalability, and maintainability emerging as critical side-effects [E5,E9,E12,E16]. Technical debt also slowed the development, as "*maintenance tasks to allow new features to be implemented at some point become rampant*" [E4].

Consequently, delays occurred [E4,E6,E9,E14,E17], as teams lost time fixing the accumulated debt. Besides time loss, dealing with technical debt [E1,E6,E9,E12,E17] also caused significant cost increases, as "*fixing debt is often not accounted for in the budget*" [E17]. Technical debt may therefore also contribute to a deteriorating customer relationship [E2,E14,E16]. Consequently, technical debt was considered a major concern by developers as well as Product Owners and managers [E2,E4,E6,E12,E17,E18, E19], particularly due to its "*medium to long-term implications*" [E19] if not addressed.

Volatile and Unclear Requirements caused incoherent products with "*low functional consistency*" [E19], forming a "*patchwork carpet of features*" [E7]. With changing requirements, the experts also reported that technical errors were induced [E2,E4, E11,E12,E16,E17], as previous work needs to be adapted to support the altered requirements. To add, the uncertainty accompanying requirement volatility "*causes mistakes to happen and creates a mess in the implementation*" [E4]. In this vein, frustration also emerges [E6,E8,E15,E16,E17], as due to volatility, "*developers start something, but are again interrupted and need to focus on something new*" [E16]. This volatility also causes a waste of resources on "*developed features that are not bulletproof*" [E9], leading to delays due to time spent on changing requirements [E6,E9,E10,E11,E16,E19].

4.2 Measures to Adapt to Issues and Sustain Value Delivery

Delivery Pressure was usually managed with three types of measures. First, experts reported that conservative estimates and smaller, manageable stories focusing on "*key aspects required for a go-live for each iteration*" [E11] helped to reduce pressure. Here, reducing the story size to a "*minimal degree of value, almost like an MVP within each story, helped us minimize the feature pressure*", reported [E16]. Second, the experts strongly involved the customer in the planning sessions and sought their agreement for conservative estimates, as it "*helped the customer understand how things get delivered and provided transparency*" [E4]. In addition, involving the customer in demos to present the value of delivered features, "*but keeping dates away from developers and instead focus on delivering quality work*" [E3] further alleviated the pressure. Moreover, "*providing transparency will put customers at ease*" [E1]. The expert added that with transparency, "*customers forgot about dates because they saw that the team was actually working on delivering the promised value*". Third, the experts emphasized a strong Scrum Master as a "*firewall*" [E9,E16] that "*protects the team*" [E3]. Especially visualizing the short- and long-term consequences of delivery pressure for value delivery "*caused a rethink on the customer side and reduced constant feature pressure*" [E16].

Lacking Customer Involvement was handled with a range of measures, such as the installation of technical consultants. These consultants, as opposed to a regular on-site customer as part of the team, sit on the customer side, "*enabling a direct connection to the customer*" [E12]. Technical consultants acted as "*fire extinguishers*" [E13] and could quickly provide necessary information, given their role to manage the customer side of development and the "*stronger relationship basis and better know-how of the customer side of the project*" [E13]. This measure was appreciated by several experts, i.e., [E5,E12,E13]. Other experts adapted to low customer involvement with proactive visits [E4,E8,E9,E16] or repeated inquiries to the customer [E2,E7,E17]. Here, asynchronous communication tools, i.e., Microsoft Teams channels and chats, proved beneficial to

"*reaching relevant contacts on the customer side*" [E2]. In addition, with a "*pac-man approach*" [E8], i.e., by "*continuously feeding the customer with something tasty*" [E3], customer involvement happened naturally, as it "*iteratively convinced them of the value that the team delivered*" [E5]. As with the issue of delivery pressure, also convincing the customer to be constantly involved by visualizing the consequences of lacking engagement is beneficial, as it helps to "*sensitize them for the importance of their role*" [E16]. As a last resort, escalations to higher levels may be an option to secure involvement, yet this step "*is rather unpleasant and implies further pressure*" [E5].

Meeting Overhead and Interruptions were managed using procedural changes, i.e., by eliminating unnecessary meetings and only focusing on the meetings that ASD prescribes. Especially jour fixes (i.e., regularly recurring, fixed meetings to discuss the current status of projects) and touchpoints that [E8] called a "*typical consulting issue*" were eliminated. For necessary meetings, experts eliminated wasteful aspects, i.e., members bringing laptops into [E3,E16] or performing work during these meetings [E4,14]. To add, Scrum Masters reduced the participating members to only those that "*can contribute substantially and not only have 30 s of participation*" [E7], so others can instead focus on coding work. [E8] defined representatives within the team, "*similar to scaled approaches where the use of representatives works well*". [E11,E14, E16] supported meetings with meeting-minute-keeping tools to ensure non-participating members could also keep themselves in the loop if desired. [E8,E15] further implemented a so-called 16th minute for daily meetings. This 16th minute implies adding a small time slot after the actual 15 min daily scrum to discuss certain issues that do not require everyone to participate, thereby saving time for other members to focus on development tasks and ensuring that the daily meeting does not significantly exceed the defined time frame. As another measure, several experts [E4,E7,E15] enforced a strict routinization of meetings to "*carve out slots for productivity*" [E5], along with a standardized agenda and a clear goal. This instilled "*a routine, which leads to efficiency in the meetings*" [E7]. Lastly, [E4,E8,E11,E16] implemented a 3-week sprint with a core focus week. Instead of spending "*two of ten working days every two weeks for sprint transition day*" [E8], a week of focus on development in between had a "*great psychological but also productivity-enhancing effect*" [E16], as it reduced meeting overhead.

Required Rework was not considered a critical issue per se, as depicted in Sect. 4.1. Still, certain measures emerged that helped to manage occurring rework. Most experts followed a good / bad / ugly rework approach, similar to the rework taxonomy by [15] to classify types of iterative rework. In this vein, good rework implies aspects that add value through beneficial adjustments toward improving the software, while bad rework entails additional efforts required due to preventable, recurring mistakes [15]. Ugly rework, meanwhile, refers to excessive amendments that cause the SD process to run out of control or that imply gold plating [15], i.e., overengineered solutions. Consequently, the experts classified rework into "*valuable improvements in the direction of the agreed vision, or rework that rather implies knocking over previous implementations, thus bad rework*" [E16]. For ugly rework, [E10] described that "*gold-plating is also undesirable, as it does not add value, takes time, and causes you to never actually finish the functionality*". To accommodate expected, beneficial rework, experts defined a rework buffer for each iteration [E3,E4,E14,E16,E19]. ASD is designed to accommodate rework, so

"*this buffer is essential during release planning since amendments will always occur somehow*" [E4]. Lastly, several experts emphasized the need for a modular and flexible architecture, thus "*setting up the product in an easily adaptable manner*" [E14] to allow demands for valuable rework to be more manageable.

Technical Debt was primarily managed through new process artifacts or technical measures. For process artifacts, experts defined a refactoring buffer that "*made up 20–30% of our capacity to look into technical aspects*" [E11]. In addition, experts created a backlog item that labels technical debt, thereby "*giving teams the psychological safety to continuously include necessary debt fixing in their planning*" [E10]. Concerning technical measures, code reviews proved particularly helpful when three developers (six-eyes-principle) inspected the code, as it prevented "*happy path testing and mutual nodding, while helping to find inconsistencies in the code*" [E11]. [E6,E18,E19] supported this identification of debt with additional tools such as CodeScene. To add, with continuous (unit) testing, potential debt can also be identified, while the system is generally "*built more modularly and more flexibly defined*" [E14]. This falls in line with several experts who emphasized modular or microservice architectures to reduce the accumulation of technical debt [E4,E5,E14]. To fix technical debt, developers applied a Boy Scout rule for clean coding, i.e., "*always leaving the software better than you found it*" [E9]. While developers adhered to this approach during regular sprints, sometimes performing a technical sprint is necessary [E8,E13,E16–18]. While such a sprint did not deliver new features per se, "*the performance gains, reduced fault likelihood, and better maintainability are convincing arguments to perform such a sprint*" [E16]. Still, in a few cases, a complete rewrite was required [E8,E9,E17]. While this is a drastic step, it eliminated "*the pain of prior implementations and allowed to get rid of inherited liabilities*" [E8], but should "*only be the ultima ratio to address technical debt*" [E9].

Volatile and Unclear Requirements could be handled with collaborative vision and expectations definitions between the team and the customer, paired with the derivation of requirements and a roadmap that aligns with the vision and expectations [E1-E3,E7,E10,E16]. Should volatility occur, assessing whether this deviation in the requirements aligns with the overall vision is essential, thereby "*challenging if this new or altered requirement is what the customer truly wants or needs*" [E5]. Here, both the Product Owner and the technical consultant can act as firewalls to assess the volatile requirements, thus "*cushioning some of the volatility*" [E13] and "*steering it in the right direction*" [E10]. These roles can also help to clarify vague and ambiguous requirements [E1,E12,E13,E16]. The experts also emphasized continuous prioritization of requirements to ascertain which "*needs are truly of high importance, and if a shift in the requirements falls in this category*" [E14]. Lastly, several experts incorporated a scope change buffer to "*accommodate things that emerge from this volatility, which we sometimes called 'surprise of the quarter'* " [E16]. This buffer allowed to capture some volatility without compromising the sprint goal [E4,E5,E11,E16]. Still, if the buffer cannot absorb the volatility, "*postponing it to the next sprint is also an option*" [E10].

Table 3. Applied measures to adapt to examined issues and sustain value delivery

Key Issue	Measure Type	Measure	Expert IDs
Delivery Pressure	Customer Approach	Demo Sessions to Show Value	E1, E3, E4, E8, E16
		Involve Customers in Planning Stage	E1, E2, E4, E13
	Procedural Changes	Inter-Sprint-Workload Balancing	E8, E10
		Workload Balancing within Team	E2, E17
	Req. Eng. Practices	Conservative Estimates	E5, E6, E8, E11, E12, E19
		Cut Small Stories	E3, E5, E16, E17
		Prioritize Key Requirements	E4, E11, E19
	Role Installation	Scrum Master as a Firewall	E3, E5, E4, E8, E10, E16
Lacking Customer Involvement	Customer Approach	Escalation to Higher Levels	E5, E10, E17
		Pac-Man Approach	E2, E3, E5, E8, E9, E19
		Proactive Customer Visits	E4, E8, E9, E16
		Repeated Inquiry	E2, E7, E17
		Visualize Consequences	E8, E9, E16, E17
	Req. Eng. Practices	Joint User Story Mapping	E1, E8, E19
	Role Installation	Implement an On-Site Customer	E8, E12, E16
		Implement a Technical Consultant	E1, E5, E12, E13, E16
	Tool Support	Asynchronous Communication Tools	E2, E6, E16, E19
Meeting Overhead and Interruptions	Procedural Changes	16^{th} Minute added to Daily Meeting	E8, E15
		3-Week Sprint with Core Focus Week	E4, E8, E11, E16
		Eliminate Unnecessary Meetings	E3, E5, E7, E11, E15, E16
		Eliminate Wasteful Meeting Elements	E1, E3, E4, E7, E14, E15

(*continued*)

Table 3. (*continued*)

Key Issue	Measure Type	Measure	Expert IDs
		Minimize Number of Participants	E5, E7, E8, E9, E17
		Routinization of Meetings	E4, E5, E7, E15
		Set up Fixed Agenda with Goal	E1, E3, E7, E9
	Role Installation	Install a Meeting Chair as Facilitator	E1, E3, E7, E14
	Tool Support	Asynchronous Communication Tools	E11, E14, E16
Required Rework	Process Artifacts	Define Rework Buffer	E3, E4, E14, E16, E19
		Good / Bad / Ugly Rework Approach	E2, E7, E8, E10, E13, E16, E17
	Role Installation	Involvement of Design Experts	E5, E8, E9, E14
	Technical Measures	Modular and Micro-Service Architecture	E4, E9, E14, E16, E17
Technical Debt	Process Artifacts	Dedicated Refactoring Buffer	E4, E8, E9, E10, E11, E12, E13, E16, E18, E19
		Technical Debt Backlog Item	E9, E10, E16, E17, E18
	Role Installation	Involve Architecture Experts	E4, E9
	Technical Measures	Boy Scout Rule for Clean Coding	E8, E9, E10, E14
		Code Guidelines	E2, E6, E9, E12, E16
		Code Reviews	E5, E10, E11, E19
		Continuous Testing / Unit Testing	E1, E10, E12, E14, E17
		Modular Architecture	E4, E5, E14
		Rebuild from Scratch	E8, E9, E17
		Technical Sprint	E8, E13, E16, E17, E18
	Tool Support	AI Code Analysis Tools and CodeScene	E5, E6, E11, E18

(*continued*)

Table 3. (*continued*)

Key Issue	Measure Type	Measure	Expert IDs
Volatile and Unclear Requirements	Process Artifacts	Scope Change Buffer	E1, E4, E5, E10, E11, E16, E19
	Req. Eng. Practices	Check Vision-Requirement-Alignment	E5, E7, E11, E12, E16
		Continuous Prioritization	E4, E11, E14, E18, E19
		Joint Requirements / Roadmap Definition	E1, E2, E3, E6, E9, E10, E14, E17
		Joint Vision and Expectation Definition	E1, E3, E5, E7, E12, E16
	Role Installation	Product Owner as a Firewall	E1, E4, E10, E14
		Technical Consultant as a Firewall	E1, E12, E13, E16, E17

5 Discussion

5.1 Key Findings and Future Research Opportunities

In this study, we examined the value-reducing consequences of six key issues in ASD and how practitioners adapt to them to sustain value delivery, which was less regarded in prior works [1, 24, 28]. Our study addresses this gap by examining the consequences of these issues if not mitigated, as illustrated in *RQ1*. In addition, a set of actionable measures to adapt to key issues was lacking. Here, our study contributes a collection of 48 measures that ASD teams can employ to enable continuous value delivery, see *RQ2*.

Concerning *RQ1*, reduced product quality and capabilities, security concerns, delays, inefficient development, and wasted effort were reported as the main consequences that affect the delivery of value in ASD. While all issues seemed problematic in some way, *lacking customer involvement*, *delivery pressure*, *technical debt*, *meeting overhead,* and *volatile requirements* appeared particularly influential. This aligns with issues identified in [1, 10, 19, 23, 24], while we extend these prior findings by detailing their specific value-reducing consequences. Consequences such as security issues induced by technical debt and delivery pressure, product-requirement-misalignments due to low customer involvement, and volatile requirements causing incoherent features suggest that adapting to these issues is essential to uphold the promise of ASD for continuous delivery of valuable software. Our work also indicates that the abovementioned issues are the key concerns that affect ASD's value proposition, except *rework*, as explained next.

As such, our work shows the need for a differentiated perception of the importance of rework for value creation in ASD research. While rework was seen as a consequence of various issues in [24] with negative implications such as increased costs [19, 30, 37], almost all experts saw rework as a "*value-enhancement with additional time needed as*

the cost involved, which is, however, necessary for a better product" [E19] and "*the only way to get value*" [E3]. This seems logical, as rework is embedded in the key XP practice of refactoring [38], making it an integral aspect of ASD. Still, some types of rework are less desirable, as apparent with the good / bad / ugly classification for rework suggested by the experts, which aligns with the rework taxonomy proposed by [15]. Overall, our study highlights rework as a key ingredient of ASD to achieve value in practice, which was somewhat misrepresented in [24], while it also emphasizes the importance of evaluating the nature of the rework that should be performed. Hence, our findings can help to resolve differing perspectives on rework between research and practice by underlining its importance for the continuous delivery of value with ASD.

Concerning *RQ2*, we present a collection of 48 measures that the experts employed to adapt to key issues encountered in ASD, which features technical measures, different ways to approach customers, or adaptations of the development process, together with new artifacts and roles. This set can assist practitioners in dealing with difficult situations that could negatively affect the delivery of value and offer strategies to improve the ASD process, thereby helping to address key issues and avoid their negative consequences from the outset. In this vein, we extend prior knowledge with insights into how to adapt to typical issues that affect value delivery, which was not addressed by [1, 24]. Additionally, while measures to adapt to certain issues were previously scattered across the literature [10, 19, 23], this study provides an empirically derived, consolidated collection. Based on this set, future research can evaluate how these measures may enhance ASD frameworks and validate their effectiveness in case studies. Research could also assess how combinations of measures stimulate each other to improve value delivery, or determine the impact of these adaptations on development success, similar to [36].

To add, our work suggests reassessing how tailoring is done when adapting to critical issues. The experts tailored ASD with method engineering [7], i.e., "*micro-level tailoring of development practices at a finer level of granularity*" [16], such as adding a 16th minute to daily meetings or following the Boy Scout rule for clean coding to reduce technical debt. Tailoring, however, is often considered from a macro perspective, i.e., if a practice is adopted or omitted [3, 36], and how these granular adaptations affect the development success [36]. In contrast, this study emphasizes the need to examine how small, nuanced adaptations improve value creation and project success. Early research on ASD customization also shows the importance of "*micro-level tailoring*" [16] of ASD practices, which may thus be further examined in research on method tailoring.

The measures can also help practitioners to better address anti-patterns that affect ASD [14], i.e., lacking client feedback due to "*long or non-existent feedback loops*" or disruptions during development, which stem from the customer side [14]. With Eloranta et al. [14] making several recommendations for various anti-patterns, our work can enrich their suggestions with different approaches to help mitigate issues in the SD process and ensure value delivery, even when facing certain problematic anti-patterns.

Lastly, resilience engineering (RE) [22] to improve the performance of ASD when facing issues provides a novel perspective for research. Applying RE in the context of ASD seems promising, as it serves to identify strategies "*to help resolve problems so that software development can progress*" [22], which aligns with the measures identified in our study to sustain value delivery. Recent work by Lopez et al. [22] shows RE as a

useful approach to detect issues and find ways of adaptation or compensation to enable resilient performance in software engineering, which also seems applicable in ASD. In line with [22], we encourage further investigations using the RE technique to develop strategies to improve ASD use. Our findings can inform these future inquiries.

In sum, our study offers actionable measures to address common issues in ASD practice, emphasizes the importance of rework to create value, and encourages scholars to investigate the role of micro-tailoring to maximize value creation and project success, which can also be enhanced with studies using the RE technique in the context of ASD.

5.2 Limitations

We discuss potential threats to validity along the dimensions by Runeson and Höst [31]. For *construct validity*, the experts may have different interpretations of the investigated issues. This can result in responses that do not accurately reflect the phenomena under study. To avoid these problems, misunderstandings related to the issues were clarified during the interviews. Further unclarities identified during the analysis were resolved with a report sent to five participants, as described in Sect. 3. We also ensured consistent questions with an interview protocol, which was refined after three interviews to prevent unclarities. To add, as an inherent limitation of our research approach, we cannot confirm if the suggested measures had the intended effects, despite specifically inquiring about measures that have proven beneficial. Given the consistency in the suggested measures, however, the collection offers valuable adaptations to common issues. Concerning *external validity* threats, the interview sample mostly consists of experts from the DACH-region, while ICT services and IT consulting bear a similarly large proportion. Thus, a regional and industry bias may influence generalizability. To reduce these concerns, we interviewed experts from other regions and industries, finding corresponding statements on issues and measures, i.e., the Pac-Man approach or the classification of good-bad-ugly rework. To add, after ~ 10 interviews, we observed answers consistent with previous interviews, indicating a certain level of saturation. We also acknowledge an inherent limitation of our research approach, as it cannot capture the details of the context in which practitioners encountered issues or employed measures to address them. Depending on the context, some measures may thus be more or less effective, while certain issues can be more problematic compared to other settings. Also, *internal validity* concerns may arise, which relate to the value-diminishing consequences. Different issues may have similar consequences, which can make it difficult to distinguish the origin of a certain value-reducing consequence. We addressed this concern by cross-checking whether other issues had similar consequences, which are therefore explicitly stated in multiple issue categories in the tables in the results section. We also ensured that a large selection of issues was examined to limit *internal validity* concerns. Lastly, for *reliability* concerns related to data collection, analysis, and reporting, we adhered to the guidelines by [6, 21, 27]. During the development of the interview protocol, the transcription, and the analysis, both authors were involved to avoid bias. Especially the analysis was performed separately, though, to ensure an unbiased interpretation. By reaching out to interviewees to avoid misinterpretations, as illustrated in Sect. 3, we further enhanced the *reliability* of our work. Despite potential limitations, our work can provide meaningful insights to support a sustained value delivery in ASD.

6 Concluding Remarks

The core proposition of ASD to continuously deliver valuable software products [4, 34] is often hampered by several issues [1, 24]. In this study, we interviewed 19 ASD experts to examine how six issues impede value delivery and to uncover how practitioners adapt to them. 34 consequences were identified, which include reduced product quality, incoherent products, delivery delays, and wasted efforts, among other consequences. To mitigate these issues, we derived a collection of 48 measures that can help ASD practitioners to sustain value delivery. Our work underlines the key role of rework for value creation, while also encouraging future studies on the benefits of micro-tailoring and the RE technique to improve the application of ASD. In sum, this study contributes helpful approaches to ensure ASD's key value proposition is upheld in daily practice.

References

1. Alahyari, H., Berntsson Svensson, R., Gorschek, T.: A study of value in agile software development organizations. J. Syst. Soft. **125**, 271–288 (2017)
2. Alami, A., Krancher, O.: How scrum adds value to achieving software quality? Empir. Softw. Eng. **27** (2022)
3. Baham, C., Hirschheim, R.: Issues, challenges, and a proposed theoretical core of agile software development research. Inf. Syst. J. **32**, 103–129 (2022)
4. Beck, K., et al: Manifesto for agile software development. (2001). https://agilemanifesto.org. Accessed 22 Jan 2025
5. Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Grünbacher, P.: Value-Based Software Engineering. Springer, Berlin, Heidelberg (2006)
6. Bogner, A., Littig, B., Menz, W.: Interviewing Experts. Palgrave Macmillan London (2009)
7. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Inf. Soft. Technol. **38**, 275–280 (1996)
8. Campanelli, A.S., Parreiras, F.S.: Agile methods tailoring – a systematic literature review. J. Syst. Soft. **110**, 85–100 (2015)
9. Conboy, K.: Agility from first principles: reconstructing the concept of agility in information systems development. Inf. Syst. Res. **20**, 329–354 (2009)
10. Dasanayake, S., Aaramaa, S., Markkula, J., Oivo, M.: Impact of requirements volatility on software architecture: how do software teams keep up with ever-changing requirements? J. Softw. Evol. Process **31**, 1–19 (2019)
11. Digital.ai: 17th Annual State of Agile Report. (2024). https://digital.ai/resource-center/analyst-reports/state-of-agile-report/. Accessed 22 Jan 2025
12. Dingsøyr, T., Nerur, S., Balijepally, V., Moe, N.B.: A decade of agile methodologies: towards explaining agile software development. J. Syst. Soft. **85**, 1213–1221 (2012)
13. Elbanna, A.R.: Identifying the risks associated with agile software development: an empirical investigation. In: Proceedings of the 8th Mediterranean Conference on Information Systems (2014)
14. Eloranta, V.-P., Koskimies, K., Mikkonen, T.: Exploring ScrumBut—an empirical study of scrum anti-patterns. Inf. Soft. Technol. **74**, 194–203 (2016)
15. Fairley, R.E., Willshire, M.J.: Iterative rework: the good, the bad, and the ugly. Computer **38**, 34–41 (2005)
16. Fitzgerald, B., Hartnett, G., Conboy, K.: Customising agile methods to software practices at Intel Shannon. Eur. J. Inf. Syst. **15**, 200–213 (2006)

17. Hannay, J.E., Benestad, H.C.: Perceived productivity threats in large agile development projects. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–10. ACM (2010). https://doi.org/10.1145/1852786.1852806
18. Highsmith, J.: Agile Software Development Ecosystems. Addison-Wesley, Boston (2002)
19. Hoda, R., Noble, J., Marshall, S.: The impact of inadequate customer collaboration on self-organizing agile teams. Inf. Soft. Technol. **53**, 521–534 (2011)
20. Kitchenham, B., Pfleeger, S.L.: Principles of survey research. ACM SIGSOFT Soft. Eng. Notes **27**, 17–20 (2002)
21. Kuckartz, U.: Qualitative Inhaltsanalyse: Methoden, Praxis, Computerunterstützung. Beltz Juventa, Weinheim, Basel (2018)
22. Lopez, T., et al.: Accounting for socio-technical resilience in software engineering. In: 2023 IEEE/ACM 16th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE), pp. 31–36 (2023). https://doi.org/10.1109/CHASE58964.2023.00012
23. Martini, A., Bosch, J., Chaudron, M.: Investigating architectural technical debt accumulation and refactoring over time: a multiple-case study. Inf. Soft. Technol. **67**, 237–253 (2015)
24. Meckenstock, J.-N.: Shedding light on the dark side – a systematic literature review of the issues in agile software development methodology use. J. Syst. Soft. **211** (2024)
25. Meckenstock, J.-N., Hirschlein, N., Schlauderer, S., Overhage, S.: The business value of agile software development: results from a systematic literature review. In: ECIS 2022 Proceedings (2022)
26. Meckenstock, J.-N., Schlauderer, S., Overhage, S.: How do individual social agile practices influence the development success? an exploratory study. In: Wirtschaftsinformatik 2022 Proceedings (2022)
27. Myers, M.D., Newman, M.: The qualitative interview in IS research: examining the craft. Inf. Organ. **17**, 2–26 (2007)
28. Petersen, K., Wohlin, C.: A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. J. Syst. Soft. **82**, 1479–1490 (2009)
29. Racheva, Z., Daneva, M., Sikkel, K., Buglione, L.: Business value is not only dollars – results from case study research on agile software projects. In: PROFES 2010, pp. 131–145. Springer (2010). https://doi.org/10.1007/978-3-642-13792-1_12
30. Ramesh, B., Cao, L., Baskerville, R.: Agile requirements engineering practices and challenges: an empirical study. Inf. Syst. J. **20**, 449–480 (2010)
31. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. **14**, 131–164 (2008)
32. Saldaña, J.: The Coding Manual for Qualitative Researchers. SAGE Publications, Thousand Oaks (2021)
33. Schryen, G.: Revisiting IS business value research: what we already know, what we still need to know, and how we can get there. Eur. J. Inf. Syst. **22**, 139–169 (2013)
34. Schwaber, K., Sutherland, J.: The scrum guide. Scrum Alliance. (2020). https://scrumguides.org/scrum-guide.html. Accessed 22 Jan 2025
35. Seaman, C.B.: Qualitative methods in empirical studies of software engineering. IEEE Trans. Software Eng. **25**, 557–572 (1999)
36. Tripp, J., Armstrong, D.J.: Agile methodologies: organizational adoption motives, tailoring, and performance. J. Comput. Inf. Syst. **58**, 170–179 (2016)
37. van Waardenburg, G., van Vliet, H.: When agile meets the enterprise. Inf. Soft. Technol. **55**, 2154–2171 (2013)

38. Wells, D.: Extreme programming: a gentle introduction (2001). http://www.extremeprogramming.org. Accessed 6 Mar 2025
39. Wiesche, M.: Interruptions in agile software development teams. Proj. Manag. J. **52**, 210–222 (2021)