Accepted to the 18th European Conference on Software Architecture (ECSA'24)

Architecture Decision Records in Practice: An Action Research Study

Bardha Ahmeti, Maja Linder, Raffaela Groner[0000−0001−8744−9203], and Rebekka Wohlrab[0000−0002−5449−7900]

Dept. of Computer Science and Engineering
Chalmers | University of Gothenburg
Gothenburg, Sweden
gusahmeba@student.gu.se, guskalmas@student.gu.se,
{raffaela,wohlrab}@chalmers.se

Abstract. To establish good architectural practice and knowledge sharing during software development and maintenance, architectural design decisions need to be documented. While a lot of research has been done on documenting architectural decisions, there exist few approaches with empirical evidence on their applicability and usefulness in practice. Architecture Decision Records are a popular documentation approach in industry, but there is a lack of research focusing on how Architecture Decision Records can be introduced in different company contexts.

To tackle this shortcoming, we performed an action research study in cooperation with a company that develops a microservice-based system without proper architecture design decision documentation. We performed seven interviews to identify the challenges faced by the developers of the system. Afterward, we introduced Architecture Decision Records as a means of documentation. Over the course of three months, we observed whether this markdown-based documentation approach addresses the identified challenges. Our results show that practitioners face challenges related to the documentation culture, knowledge transfer, prioritization of information to be documented, as well as handling documentation for shared and distributed components. The first three types of challenges are well addressed by Architecture Decision Records. However, challenges arising from developing distributed systems remain open. Thus, there is a need for further research that helps to document design decisions for distributed systems effectively. We also compiled a list of lessons learned from our study. We found that cooperation among the teams was improved after the introduction of Architecture Decision Records. At the same time, the decision on where documentation is stored has a massive influence on its perceived usefulness. Practitioners should carefully consider what information to store centrally and what information to store in local repositories.

Keywords: Action Research · Architecture Decision Records · Architecture Decision Documentation · Challenges · Software Architecture.

1 Introduction

A selected software architecture influences a software system in all stages of the software life cycle. Therefore, its documentation is an important part of maintaining a system efficiently for years, since missing or inadequate documentation can delay maintenance or the onboarding process of new employees.

In particular, the documentation of architecture design decisions (ADDs) plays an important role. Each decision is based on an analysis of the advantages and disadvantages as well as a discussion of the technical options. Documenting this information helps later to understand why a system has been implemented in a certain way. It also helps to support future decisions and thus to avoid the repetition of wrong decisions.

A lot of research has examined ADDs. For example, several tools facilitate ADDs, like the Architecture Design Decision Support System (ADDSS) [4] tool and Knowledge Architect (KA) [14]. Additionally, there are several approaches to document ADDs, e.g., Architecture Decision Records (ADRs) [17, 12] or UML.

While ADRs are popular in industry [10], there is a lack of empirical evidence on how ADRs can be introduced in companies as a means of documenting ADDs. Most empirical studies focus on architectural documentation in general. For example, Rost and Robillard [7] examined the extent to which the format of architectural documentation affects comprehension. Manteuffel et al. [15] explored whether architectural decisions from previous projects can be reused in new projects. Finally, Buchgeher et al. [3] mined open-source repositories at GitHub to analyze the current use of ADRs.

However, no study addresses the challenges that practitioners face if they do not have any proper documentation approach and how introducing ADRs can overcome these challenges from a culture, tooling, and knowledge management perspective. Therefore, we address the following research questions in our work:

RQ1: What challenges do practitioners face while developing microservices without documentation of ADDs?

RQ2: To what extent are these challenges addressed when introducing ADRs as a means of documenting ADDs?

To answer our two research questions, we performed action research in cooperation with a company to investigate whether ADRs are suitable as a means of documenting ADDs in practice. The system under consideration is based on microservices and is developed and maintained by various agile teams.

Our results show that different types of challenges are more or less well addressed by ADRs. For example, challenges related to documentation culture or knowledge transfer are well addressed by ADRs. However, challenges arising from the distributed development of partially dependent components are insurmountable using only ADRs and require further development. Also, a clear definition of documentation guidelines is crucial for implementing a documentation approach.

2 Related Work

In this section, we discuss a selection of related works. Firstly, we present work that presents tools or other documentation approaches to document ADDs. Secondly, we describe studies that deal with architecture documentation in practice.
Documentation of ADDs: Several works deal with the documentation of ADDs. In contrast to our work, however, no challenges are mentioned in these works and these works do not consider microservice-based systems.

For example, Shahin et al. [20] evaluated nine different ADD models and compare different tools that work with these models. Many papers, such as Lee and Kruchten [13], Tyree and Akerman [22], Heesch et al. [23], and Kopp et al. [12], present tools, views, or models for documenting ADDs. Often, these approaches are similar to ADRs or use ADRs. Heesch et al. [23], for example, described various views on documentation, one of which is very similar to the ADRs we use. Kopp et al. [12] presented their tool for documenting ADDs with the help of ADRs and evaluate their tool with several participants.

Haselböck et al. [9] analyzed different applications of decision models in microservice-based architectures and developed their own decision models. They introduced these models as part of a technical action research process in several companies to identify relevant stakeholders and use cases for their models. In this phase, the authors also collected requirements for their models. Like our work, Haselböck et al. [9] also considered the documentation of ADDs in the context of microservice-based systems. In comparison to their work, we use action research to identify challenges that arise in such systems without proper documentation and observe how the introduction of ADRs affects them.

Alexeeva et al. [2] analyzed 96 publications covering 58 approaches to document ADDs and obtained a taxonomy for ADDs. The authors have also compiled a list of possible factors for the rare use of ADDs in practice, such as the resulting documentation overhead not being taken into account and the lack of integration in commercial tools. This list includes system-independent challenges about ADDs. In comparison, our work focuses on the challenges in the context of microservice-based systems.
Studies on Architecture Documentation in Practice: Several studies have focused on the challenges of documenting the architecture of a system or ADDs. Dasanayake et al. [6] performed a case study to examine the approaches developers use to make an ADD and what challenges they face during the decision-making. Rost et al. [18] also conducted a study in which they interviewed 147 developers from the industry. Their study aimed to explore the challenges developers face when realizing an architecture based on its documentation. In comparison to these studies, we include different groups of stakeholders, such as developers, architects, and team leaders, and examine the challenges they face without proper ADD documentation. We also analyze how those challenges can be mitigated by introducing ADRs.

Forward and Lethbridge [8] surveyed 32 developers about their documentation, with a focus on the tools used and the maintenance of the documentation. Manteuffel et al. [16] performed a case study at a company. During this study, the authors identified several challenges the developers face while they document ADDs. For instance, the developers lack tool support and guidelines for documenting ADDs. Based on their observations, a tool was developed that implements the Decision Documentation Framework [23].

Although the studies by Forward and Lethbridge [8] and Manteuffel et al. [16] presented challenges related to documentation and ADDs, they did not explicitly focus on distributed microservice-based systems. In addition, they do not include how the introduction of a documentation approach addresses these challenges.

The publication by Kleehaus and Matthes [11] is an example of a work that deals with the documentation of microservice-based systems. The authors presented various challenges that developers face when documenting such a distributed system. In comparison to this work, we focus specifically on the documentation challenges in a real-world context, the introduction of ADRs, and observations regarding to what extent the observed challenges were mitigated.

3 Architecture Decision Records

In this section, we provide some background on Architecture Decision Records and introduce the template we use in this work.

Architecture Decision Records (ADRs) were introduced by Nygard [17]. The idea is to make decisions comprehensible with the help of lightweight documentation in a Markdown language. Nygard proposes a five-part structure, starting with the *title* of the document. The *context* section summarizes the factors influencing a decision, such as technical, social, or political factors. The *decision* section describes the responses to the influencing factors from the context section. The *status* section documents whether a decision is proposed, accepted, deprecated, or superseded. Finally, the resulting *consequences* are summarized [17].

Listing 1.1 shows a simplified version of an ADR that we provided to the study participants as an initial example. The structure of this ADR is mainly based on the structure presented by Nygard. However, our template also contains a section to document other options considered and the reasoning for why they were discarded. We also included a section for tags to label an ADR. Before the introduction of ADRs at the company at which we conducted our study, we held a workshop on ADRs. At this workshop, the practitioners provided feedback that they would like to include these two sections in the ADR template.

```
# ADR 000 Using ADRs

## Context
Architecture decisions are currently not documented in a
structured and consistant way.

## Decision
We will document every architecturally significant
decisions as ADRs. They will be stored in Markdown format.

## Considered Options
```

```
--

## Status
Accepted

## Consequences
1. Team members should write an ADR and submit it for
review before implementing an approach to any architectural
decision.
2. We will have a visible history of the decisions
evolution through version control.

## Tags
Documentation, Architecture Decision
```

Listing 1.1. Simplified Architecture Decision Record.

4 Study Context

We performed our study in cooperation with a Swedish engineering and consulting company that offers engineering, design, digital, and advisory services in different areas. Two teams participated in our study, both of which applied the principles of agile software development. The teams already used tools, e.g., a wiki page, and visualizations to document parts of their system but lacked an approach to document ADDs in an effective and structured way. Both teams were responsible for developing and maintaining a microservice-based system that was used by the 19,000 employees of the company.

One team (team_{Sys}) consisted of eight employees, namely one business analyst, one tester, and seven developers/architects. Team_{Sys} was responsible for the overall architecture and platform design of the system considered in our study. The other team (team_{App}) consisted of six employees, namely one team leader, three developers, one tester, and one business analyst. Team_{App} was responsible for a few specific application services consumed by the system considered.

The teams regularly communicated architecture decisions through a developer forum, where all the teams met, presented, and discussed the decisions taken. However, the lack of centralization and accessibility of documentation led to confusion and slowed down the decision-making process.

5 Research Method

In this section, we describe the research method we used to answer our two research questions. All supplementary material can be found online [1].

Since we want to learn the challenges that practitioners face while developing microservice-based systems without proper documentation of architecture design decisions and how architecture decision records tackle these challenges as a means of documenting ADDs in practice, we selected a research method that focuses on the study context, learning, and improvement of the practice. Our study was conducted over the course of three months. We applied action research [21], which consists of five phases, namely *Diagnosing*, *Action Planning*, *Action Taking*, *Evaluation*, and *Learning* [21]. In the following, we describe our study design that implements these five phases.

Diagnosing: The first phase of action research was dedicated to identifying the problem to be addressed [21]. We combined semi-structured interviews, a survey, and participant observations to collect data in this phase and answer **RQ1** [5]. A total of seven employees from both teams took part in the interviews. We transcribed the recordings of the interviews and analyzed them with the help of open coding using descriptive codes [19]. We provide information on the codes used during our analysis at [1]. Six employees participated in our survey, and we analyzed the results using descriptive statistics. For the observations, we accompanied the participants to various daily meetings to observe the team members in their usual activities. The observations were carried out in a non-intrusive manner using notes to collect data about the daily work of the teams. We present the results of our *Diagnosing* phase in Section 6.1.

Action Planning: This phase was dedicated to planning our study based on the results from the *Diagnosing* phase [21]. Based on the result from the previous phase, we designed a plan and guidelines to introduce Architecture Decision Records (ADRs) as a means to document ADDs. The plan elaborated by us consisted of three stages. First, a workshop was held to introduce ADRs. We also discussed the structure, storage, visibility, and handling of ADRs with the participants at this workshop. Second, in collaboration with the participants, we introduced seven guidelines on how the team members should manage ADRs. We assigned each guideline a unique ID using the naming schema **G<number>** These guidelines were also published on a central wiki page and are as follows:

G1 ADRs affecting several microservices are stored in a central repository.
G2 ADRs are referred to in commit messages and user stories.
G3 ADRs are checked in with the code fragments connected to them in the corresponding repository.
G4 ADRs are reviewed by another developer and are published after a successful review.
G5 ADRs are tagged with the quality attributes, components, or features they are related to.
G6 ADRs are labeled with a status to document whether an ADR is proposed, accepted, rejected, deprecated, or superseded.
G7 New ADRs are discussed in biweekly developer forums.

Finally, the team members used ADRs in their daily work. To make this step easier, we provided the teams with an example ADR. A simplified version of this ADR is shown in Listing 1.1. During this stage, we collected data using semi-structured interviews, a survey, and observation of the study participants.

Action Taking: During this phase, the plan designed during the *Action Planning* phase was performed [21]. During this stage of our study, we spent four weeks at the company to collect data.

Evaluation: In this phase, the aim was to understand the influence of the previously performed action [21]. As already mentioned, we analyzed our data qualitatively with the help of coding.
Learning: The final phase in the cycle dealt with the lessons learned from the previous phases [21]. Based on the combination of our evaluation of the results of the *Action Taking* phase and our results from the *Diagnosing* phase, we obtained different results for this phase to answer **RQ2**. We present the results of this phase and our lessons learned in more detail in Section 6.2.

6 Findings

In this section, we present our findings and present our lessons learned. In Section 6.1, we present the challenges the teams face without proper documentation of ADDs. In Section 6.2, we present the observations we made after ADRs were used for documentation and map them to the previously identified challenges. Subsequently, we discuss and summarize our lessons learned.

6.1 Challenges Due to Insufficient Documentation (RQ1)

In this section, we focus on the challenges that practitioners face without proper ADD documentation. First, we present insights that show how current team members perceived the practice of documenting ADDs before introducing ADRs. Afterward, we present the seven challenges that practitioners face without proper documentation of ADDs.

Figures 1 and 2 show the results of the survey we performed during the *Diagnosing* phase. Both plots consist of two parts: The part on the right shows the number of participants who answered “Don’t know”. The part on the left shows the number of participants who answered the respective question. The bars in the left part of the plot grow from the center to the left to present the negative answers of the participants. To represent the positive answers the respective bar grows from the center to the right. The annotations show the percentage share of the six participants who answered the respective question positively or negatively.

In Figures 1 and 2, it is noticeable that at the beginning of our study, 83% of respondents stated that ADDs are only rarely or occasionally documented and that there are no clear guidelines for documenting ADDs. In addition, half of the survey participants are dissatisfied with the way ADDs are documented.

Based on the analysis of our qualitative data, we identified seven challenges that can be categorized into four main categories, namely **Documentation Culture**, **Shared/Distributed System Parts**, **Knowledge Transfer**, and **Prioritization**. To enable traceability, we assign each challenge a unique ID using the naming schema **C<number>**.
Documentation Culture: The first challenge is related to the problems of fostering a documentation culture in an agile team (**C1**). Figure 2 shows the dissatisfaction of the practitioners with the currently available documentation.

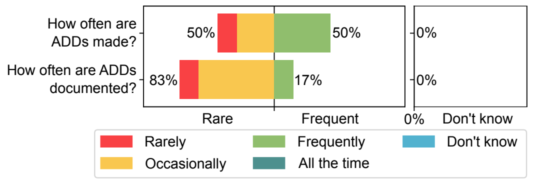

Fig. 1. Frequency of ADDs and their documentation before introducing ADRs

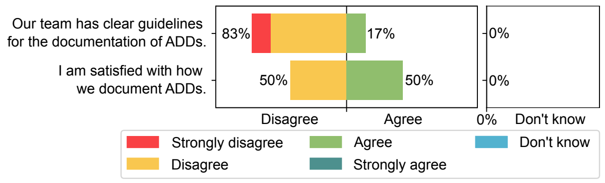

Fig. 2. Assessment of the current ADD documentation before introducing ADRs

Additionally, some team members elaborated on their dissatisfaction due to missing documentation during our interviews. Based on the participants' statements, we identified one challenge related to the documentation culture. For example, an architect from team$_{\text{Sys}}$ mentioned: "*It seems to be challenging also when you work with agile and you have to continuously deliver. It seems like stopping for documentation is a challenge for teams across the industry so yeah, documentation is hard.*"

Shared/Distributed System Parts: Different teams are involved in developing the microservice-based system at the company we cooperated with. Thus, the corresponding code is stored in different repositories. In this context, some participants mentioned that the placement of documents can affect their usefulness. One challenge relates to identifying and receiving information about ADDs that affect all or multiple components of a system (**C2**). This challenge is supported by the statement from an architect from team$_{\text{Sys}}$: "*We have a lot of different teams that are working with the different parts of our platforms. But sometimes, there is stuff that is common for all the teams, and then finding that decision and finding why we did something or haven't done something, it is hard.*"

Another challenge relates to the location where the documentation is stored because the documentation of ADDs is currently scattered across different tools and platforms (**C3**).

Knowledge Transfer: Documentation is a means of knowledge transfer. We identified two challenges arising from a lack of/insufficient documentation in knowledge transfer.

The lack of documentation leads to relying on individual team members to maintain the context and history of ADDs and recall them from memory (**C4**). Relying on the memory of individual team members as a means of documentation is not expedient. On the one hand, essential knowledge gets lost when team members leave the company. On the other hand, individual team members are burdened with a high mental load and responsibility, as they are expected to remember all decisions correctly.

Another challenge is related to onboarding new team members. Due to outdated and missing documentation, onboarding depends on meetings with other team members (**C5**). As a result, the employees' productivity decreases during the onboarding of new team members. On the one hand, senior employees are busy with onboarding tasks that could be covered by providing documentation. On the other hand, the onboarding of a new employee depends on the availability of other team members. This problem can also be seen in the statement of a developer from Team$_{\text{App}}$: "*[...] it kind of takes time from developers to share the information every time a new person starts.*"

Prioritization: Besides the issue of where the documentation should be stored, another important aspect is deciding on what should be documented. One challenge deals with missing prioritization of information to be documented (**C6**). A lack of prioritization might lead to too much documentation, which overwhelms the team members due to the resulting workload and excessive documentation.

Another challenge is partly related to the previously mentioned challenge, as it encompasses the lack of guidelines for prioritizing information to be documented (**C7**). This lack of guidelines is also reflected by the answers to our survey during the *Diagnosing* phase. As shown in Figure 2, four of the six survey participants stated that they disagree with the statement "*Our team has clear guidelines for the documentation of ADDs.*" Additionally, one participant stated that they strongly disagreed with the statement.

Answer to RQ1: The majority of the challenges we identified are independent of the developed system. For example, the lack of guidelines for documentation or the loss of expertise due to a lack of documentation are challenges that exist independently of the system architecture. Nevertheless, we identified two system-specific challenges. The first challenge is concerned with identifying and receiving information that is relevant for multiple microservices. The second one is concerned with documentation scattered across different tools.

6.2 Observations after introducing ADRs (RQ2)

After introducing ADRs as a means of documenting architectural design decisions, we observed several changes in the teams that are related to the seven challenges we presented in Section 6.1. Figure 3 shows an overview of the team

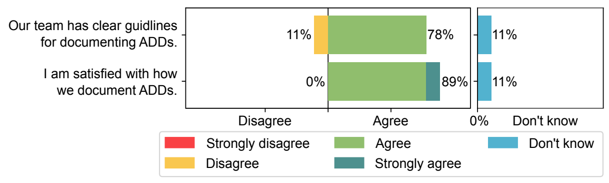

Fig. 3. Assessment of the ADD documentation using ADRs

members' perceptions after introducing ADRs. It can be seen that most participants considered their team to have clear guidelines and that they were satisfied with their documentation approach. If we compare Figure 3 with Figure 2, we can see a more positive assessment compared to before the introduction of ADRs for documentation. If we assign numerical values to the nominal scale with "strongly disagree"=1 to "strongly agree"=4 and exclude the two "don't know" responses, we see that the average agreement values also increased. The average agreement with the statement about guidelines increased from 2 to 2.9. The average agreement increased from 2.5 to 3.1.

In this section, we describe our observations and map them to their related challenges using our identifier schema: **C<number>**. In the following, we assign each observation an identifier using the naming schema **O<number>**.

In total, we gathered nine observations that show to what extent the challenges were tackled by the introduction of ADRs to document ADDs. An overview of our findings can be seen in Table 1 which summarizes the identified challenges and maps them to our observations. In this section, since we map the previously identified challenges to observations, we reuse our four main categories (**Documentation Culture**, **Shared/Distributed System Parts**, **Knowledge Transfer**, and **Prioritization**) to classify our observations.

Documentation Culture: C1 is related to the difficulties in maintaining documentation. During our study, we observed different changes related to this challenge: First, writing ADRs led to team members discussing the importance of documentation (**O1**). Team$_{\text{App}}$ introduced a review process for ADRs and integrated ADRs into its daily meetings. This also encouraged collaboration between developers in the creation of ADRs (**O2**). Furthermore, team members mentioned that ADRs were easy to create (**O3**). For example, an architect from team$_{\text{Sys}}$ mentioned "*I think it's really easy to write them actually. They are basic enough, as they should be.*"

Shared/Distributed System Parts: C2 is concerned with retrieving information about decisions that affect multiple parts of the system. To deal with this challenge, for this study we decided to document such decisions in a separate repository for all teams (cf. **G1** in Section 5). Seven of nine participants of our second survey stated that they used the documentation (**O4**). In addition to the guidelines we provided, some participants expressed the wish to integrate the documented ADRs into the rest of the system documentation (**O5**).

The practitioners proposed a tag-based system to address the challenge of scattered ADD documentation across different tools and platforms (**C3**). Their idea was that ADRs should be categorized using standardized tags and then all ADRs could be filtered using these categories (**O6**). The ADR template we used in our study already provides a section for tags, but these tags were not standardized and we did not provide a search mechanism.

Knowledge Transfer: C4 is concerned with relying on individual team members to remember and correctly recall ADDs. We observed that this challenge is eliminated by using ADRs as a means of documenting ADDs (**O7**). This is also reflected in the statement of a developer from Team$_{\text{App}}$: "*What I like most*

Table 1. Overview of the identified challenges and the associated observations.

Challenges		Observations	
ID	**Description**	**ID**	**Description**
C1	Foster and maintain a documentation culture in an agile team	**O1**	Writing ADRs sparked discussions about the importance of documentation.
		O2	ADRs are perceived as encouraging collaboration.
		O3	ADRs are perceived as an easy way to document ADDs.
C2	Identifying and receiving information about ADDs that affect all or multiple components of a system	**O4**	Participants used the documentation of overarching ADRs.
		O5	Participants want to include ADRs in the system documentation.
C3	The documentation of ADDs is scattered across different tools and platforms.	**O6**	Participants suggest standardized tags to filter ADRs, independently of the repositories they are stored in.
C4	Relying on individual team members to maintain the context and history of ADDs	**O7**	By using ADRs, team members no longer need to memorize ADDs.
C5	Onboarding depends on meetings with other team members	**O8**	Participants rate it as likely that they will integrate ADRs into the onboarding process.
C6	Lack of prioritization of the information to be documented	**O9**	Provided guidelines helped the participants, but they need further improvement.
C7	Lack of guidelines for prioritizing		

about ADRs is that I do not need to remember these things further, I can write them down here and now and then be sure that I won't lose this information."

The timeframe of our study was too short to determine whether the use of ADRs makes the onboarding process less dependent on meetings with experienced team members (**C5**). In our second survey, we asked participants using a 5-point scale (1="very unlikely" to 5="very likely") about the likeliness of including ADRs in the onboarding process. Two participants rated the likeliness with 3, five participants rated it with 4, and two rated the likeliness with 5. The rating indicates that the participants are very positive about integrating ADRs into the onboarding process (**O8**).

Prioritization: In terms of prioritization, we were able to identify two challenges. Firstly, what should be documented (**C6**), and secondly, what guidelines should be applied for prioritization (**C7**). Related to these two challenges, we observed that the seven guidelines we provided are already helping the team members. Nevertheless, the participants noted that the guidelines need to be further improved (**O9**). For example, some team members struggled to decide whether a change should be considered as an ADD and thus needs to be documented using an ADR. This is also reflected in the statement of an architect from team$_{\text{Sys}}$: "*The hardest part might be to actually decide whether it's a change that is made, whether it's to be considered as an ADR, or if it's just part of normal maintenance.*"

Answer to RQ2: Our results show that challenges related to system-independent categories like documentation culture or knowledge transfer are addressed well using ADRs. Challenges that practitioners face due to the distributed nature of the microservice-based system are not addressed and need further research. To tackle this issue, the participants made some suggestions like implementing standardized tags and a filtering mechanism that allows them to search for ADRs across all tools and repositories.

6.3 Discussion and Lessons Learned

Based on our findings, we have compiled a list of lessons learned, which we discuss and summarize in this section.

Documentation Culture: We have learned that introducing ADRs has led to stronger cooperation among the teams (O2).

Previously, ADDs were discussed and communicated in wiki pages and discussions in development forums or via teams. After the introduction of ADRs, the developers wrote ADRs cooperatively. Additionally, there was a stronger focus and more discussions in the teams on how, where, and what should be documented. Despite these positive effects, some effort is needed to integrate ADRs into existing processes. For example, some developers had difficulties integrating the creation of ADRs into their daily workflow.

It is not always the case that the documentation of decisions strengthens cooperation within a team. For example, Dasanayake's [6] survey shows that cooperation depends on the respective teams. This is also shown by the case study of Forward and Lethbridge [8], in which decisions are often made in cooperation, regardless of whether they are documented.

Therefore, we assume that this lesson learned is independent of the method of documentation used. However, we expect the corporate culture and social dynamics within a development team to have a strong influence. The teams in our study use agile software development, which might have also influenced this lesson learned.

Guidelines and Prioritization: We have learned that clear guidelines on the content to be documented are crucial for implementing a documentation approach successfully (C7).

During our study, practitioners sometimes struggled to decide whether a decision was an architecturally significant decision and whether they should create an ADR. Furthermore, practitioners need additional guidelines to help them prioritize the information to be documented, as they only have limited time available due to the short development cycles in agile software development. Our participants have prioritized their documentation based on the complexity of the logic involved, the impact of changes on the system, and the number of developers or teams affected. Some practitioners suggested prioritizing based on the criticality of the system or component or the frequency of changes.

The problem of a lack of guidelines for documentation is also mentioned by the participants in the evaluation of the documentation framework by Manteuffel et al. [16]. In this work, the participants state that they document “major” decisions without a clear definition of what “major” means in a given context.

Our seven guidelines from Section 5 are context-independent. Thus, they can applied to other teams without any adjustments. However, we expect the two shortcomings related to explicitness and prioritization mentioned above to be context-independent. Thus, these shortcomings might also occur in other teams.

Shared and Distributed System Parts: We have learned that the storage location of the documentation has an extreme impact on its usefulness and accessibility (C3).

On the one hand, the documentation of shared system components should be accessible to all teams for which the corresponding ADRs are relevant. On the other hand, the practitioners prefer to store the documentation with the corresponding system part/code. One developer commented on this way of organizing documentation. They stated that the documentation for a service might be stored in its repository because the developers do not know that their ADD affects several other services as well. Thus, developers of a service need to know which services depend on their service and whether their ADD affects other services to decide where to store the documentation.

One approach is to store the documentation locally with the automated publication of the ADRs on a central wiki page or similar. In our study, the practitioners also noted that ADRs should be provided with standardized tags. These tags should be used to filter the ADRs and identify relevant ADRs across all platforms and tools used.

Regardless of whether a system is distributed or not, practitioners report that important information is often scattered strongly [18]. For example, developers use different tools for documentation [8]. Practitioners also mention difficulties in navigating existing documentation [18] or revisiting the design rationale [6]. In the Forward and Lethbridge survey [8], participants mentioned that centralized documentation beyond individual projects would help them.

This lesson applies to all distributed systems, especially when ADDs affect multiple system components. However, we also see in other works that teams have problems organizing documentation in a consistent and easy-to-find way, regardless of the system developed. Whether the solution proposed by the practitioners solves this issue needs to be explored in more detail.

7 Threats to Validity

In the following, we discuss threats to the validity of our study.

Conclusion validity: Our results might be only a small subset of possible challenges and observations since we performed our study in the context of one company and used mainly data collection methods that produce qualitative data.

However, with the help of action research in the context of one specific company, we can directly observe the impact of introducing ADRs to document ADDs in practice. Additionally, the qualitative data we obtained provides us with more insights into the challenges that practitioners face without proper documentation and what changes arise in the company due to introducing a systematic way to document ADDs.
Internal validity: Interviewing participants can always lead to incorrect data being collected due to the interviewee's bias or inaccuracies. Therefore, we applied multiple methods, as this allows us to triangulate data from interviews, surveys, and participant observation.

The interviews were coded by two researchers independently and their results were subsequently compared to avoid researcher bias and mistakes.
Construct validity: We performed pilot interviews with two students and a software developer to test the interview procedure and identify ambiguous questions.

The implementation of data collection based on observations is always prone to participants behaving differently under observation than in real life. We tried to observe the participants as non-intrusively as possible. However, we cannot say to what extent our presence influenced the participants' behavior.

Terms like "decision" can be interpreted in different ways by different participants. To ensure that a common meaning is used, those constructs were briefly discussed at the beginning of the interviews, when introducing the topic.
External validity: We cannot say to what extent our results apply to other companies. However, we expect the identified challenges to apply to other companies with a similar working setup as the company in our study. There might be further challenges, e.g., related to the storage and distribution of ADD documentation, the size of the development teams, or the complexity of the microservice-based system. The behavior of the participants, on the other hand, may differ from our observations even in an identical company, as this depends on various factors such as team dynamics, personality, and company culture.

We used ADRs for the documentation of ADDs. We cannot say to what extent this influenced our results. We assume that the use of other documentation approaches will produce slightly different results. For example, if our study is repeated using a documentation approach that provides dedicated tool support, the usability of such a tool is an additional, decisive factor influencing the results.
Reliability: We published our study material at [1] to increase the reliability of our study. However, we cannot publish the raw data due to confidentiality. It should be noted, that our results are based on experience in a company. Thus, we cannot guarantee that researchers produce the same results if they repeat our study in another company.

8 Conclusion

This research was motivated by a lack of studies on how introducing ADRs can overcome documentation challenges from a culture, tooling, and knowledge management perspective. To close this gap, this action research study has examined the impact of ADRs on the challenges associated with documentation practices during the distributed development of a microservice-based system.

Our results show that challenges can be divided into different categories. Challenges that belong to categories that can also occur in non-distributed systems, such as knowledge transfer, are already well addressed by ADRs in combination with clear guidelines for documentation. However, for challenges arising from the distributed system developed, there is currently no suitable solution. This lack of support in documenting ADDs for distributed systems is also evident from our lessons learned. Therefore, we see a need for future work, especially on the effective management of ADD documentation in distributed systems.

9 Data Availability

We published our study material consisting of two questionnaires, our interview guides, and information on the codes used during our analysis at [1]. We cannot publish the collected raw data due to confidentiality.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

1. Ahmeti, B., Linder, M., Groner, R., Wohlrab, R.: Supplementary material. `https://doi.org/10.5281/zenodo.11635100` (2024)
2. Alexeeva, Z., Perez-Palacin, D., Mirandola, R.: Design decision documentation: A literature overview. In: Tekinerdogan, B., Zdun, U., Babar, A. (eds.) Software Architecture. pp. 84–101. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-48992-6˙6
3. Buchgeher, G., Schöberl, S., Geist, V., Dorninger, B., Haindl, P., Weinreich, R.: Using architecture decision records in open source projects—an MSR study on GitHub. IEEE Access **11**, 63725–63740 (2023). https://doi.org/10.1109/ACCESS.2023.3287654
4. Capilla, R., Nava, F., Montes, J., Carrillo, C., et al.: ADDSS: architecture design decision support system tool (2010)
5. Clark, V.L.P., Ivankova, N.V.: Mixed methods research: A guide to the field, vol. 3. Sage publications (2015)
6. Dasanayake, S., Markkula, J., Aaramaa, S., Oivo, M.: Software architecture decision-making practices and challenges: An industrial case study. In: Proceedings of the 2015 24th Australasian Software Engineering Conference. pp. 88–97 (2015). https://doi.org/10.1109/ASWEC.2015.20
7. Ernst, N.A., Robillard, M.P.: A study of documentation for software architecture. Empirical Software Engineering **28**(5), 122 (2023)

8. Forward, A., Lethbridge, T.C.: The relevance of software documentation, tools and technologies: a survey. In: Proceedings of the 2002 ACM Symposium on Document Engineering. p. 26–33. DocEng '02, Association for Computing Machinery, New York, NY, USA (2002). https://doi.org/10.1145/585058.585065
9. Haselböck, S., Weinreich, R., Buchgeher, G.: Decision models for microservices: Design areas, stakeholders, use cases, and requirements. In: Lopes, A., de Lemos, R. (eds.) Software Architecture. pp. 155–170. Springer International Publishing, Cham (2017)
10. Keeling, M.: The psychology of architecture decision records. IEEE Software **39**(6), 114–117 (2022). https://doi.org/10.1109/MS.2022.3198195
11. Kleehaus, M., Matthes, F.: Challenges in documenting microservice-based it landscape: A survey from an enterprise architecture management perspective. In: Proceedings of the 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC). pp. 11–20 (2019). https://doi.org/10.1109/EDOC.2019.00012
12. Kopp, O., Armbruster, A., Zimmermann, O.: Markdown architectural decision records: Format and tool support. In: ZEUS. pp. 55–62 (2018)
13. Lee, L., Kruchten, P.: A tool to visualize architectural design decisions. In: Becker, S., Plasil, F., Reussner, R. (eds.) Quality of Software Architectures. Models and Architectures. pp. 43–54. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
14. Liang, P., Jansen, A., Avgeriou, P.: Knowledge Architect: A Tool Suite for Managing Software Architecture Knowledge. University of Groningen, Johann Bernoulli Institute for Mathematics and Computer Science (2009), relation: http://www.rug.nl/informatica/organisatie/overorganisatie/iwi Rights: University of Groningen, Research Institute for Mathematics and Computing Science (IWI)
15. Manteuffel, C., Avgeriou, P., Hamberg, R.: An exploratory case study on reusing architecture decisions in software-intensive system projects. Journal of Systems and Software **144**, 60–83 (2018). https://doi.org/10.1016/j.jss.2018.05.064
16. Manteuffel, C., Tofan, D., Koziolek, H., Goldschmidt, T., Avgeriou, P.: Industrial implementation of a documentation framework for architectural decisions. In: Proceedings of the 2014 IEEE/IFIP Conference on Software Architecture. pp. 225–234. IEEE (2014)
17. Nygard, M.: Documenting architecture decisions. `https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions` (2011), online; accessed 12-April-2024
18. Rost, D., Naab, M., Lima, C., von Flach Garcia Chavez, C.: Software architecture documentation for developers: A survey. In: Drira, K. (ed.) Software Architecture. pp. 72–88. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
19. Saldaña, J.: The coding manual for qualitative researchers (2013)
20. Shahin, M., Liang, P., Khayyambashi, M.R.: Architectural design decision: Existing models and tools. In: Proceedings of the 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. pp. 293–296 (2009). https://doi.org/10.1109/WICSA.2009.5290823
21. Staron, M.: Action research in software engineering. Springer (2020)
22. Tyree, J., Akerman, A.: Architecture decisions: demystifying architecture. IEEE Software **22**(2), 19–27 (2005). https://doi.org/10.1109/MS.2005.27
23. van Heesch, U., Avgeriou, P., Hilliard, R.: A documentation framework for architecture decisions. Journal of Systems and Software **85**(4), 795–820 (2012). https://doi.org/10.1016/j.jss.2011.10.017