Received 1 June 2023, accepted 7 June 2023, date of publication 19 June 2023, date of current version 28 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3287654

RESEARCH ARTICLE

Using Architecture Decision Records in Open Source Projects—An MSR Study on GitHub

GEORG BUCHGEHER[1], STEFAN SCHÖBERL[2], VERENA GEIST[2], BERNHARD DORNINGER[2], PHILIPP HAINDL[3], AND RAINER WEINREICH[4]

[1]karriere.at GmbH, 4020 Linz, Austria
[2]Software Competence Center Hagenberg GmbH, 4232 Hagenberg, Austria
[3]Department of Computer Science and Security, St. Pölten University of Applied Sciences, 3100 St. Pölten, Austria
[4]Department of Business Informatics—Software Engineering, Johannes Kepler University Linz, 4040 Linz, Austria

Corresponding author: Verena Geist (verena.geist@scch.at)

The research reported in this article has been funded in part by the Austrian Research Promotion Agency (FFG) BRIDGE project AK-Graph under Grant no. 883718; in part by the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK), the Federal Ministry for Labour and Economy (BMAW), and the State of Upper Austria in the frame of the Software Competence Center Hagenberg (SCCH) competence center INTEGRATE under Grant no. 892418 in the COMET-Competence Centers for Excellent Technologies Programme managed by Austrian Research Promotion Agency FFG; and in part by the Interreg Österreich-Bayern 2014–2020 Programme under Grant AB292.

ABSTRACT Architecture decision records (ADRs) have been proposed as a resource-efficient means for capturing architectural design decisions (ADDs), and have received attention not only from researchers but also from practitioners. We conducted a mining software repositories (MSR) study, in which we analyzed the use of ADRs in open source repositories at GitHub. Our results show that the adoption of ADRs is still low, although the number of repositories using ADRs is increasing every year. About 50% of all repositories with ADRs contain just one to five ADRs suggesting that the concept has been tried but not yet definitively adopted. In repositories that use ADRs more systematically, we observed that recording decisions is a team activity conducted by two or more users over a longer period of time. In most repositories the template proposed by Michael Nygrad is used. We, finally, provide an interpretation of the obtained results and discuss open future research challenges by elaborating on implications of the study's findings as well as on recommendations on how to further increase the adoption of ADRs.

INDEX TERMS Architecture decision records, mining software repositories, secondary study, GitHub, open source projects, software architecture, software architecture knowledge management.

I. INTRODUCTION

In 2004, Bosch [1] proposed to extend the view on software architecture beyond architectural structures and also to consider architectural design decisions (ADDs) as first class entities. ADDs seek to capture the rationale behind architecture solution structures, along with design knowledge (i.e., the pros and cons of a decision, the considered design alternatives, and the consequences resulting from a decision). This knowledge would otherwise remain implicit knowledge in the minds of software architects and is subject of architecture knowledge vaporization, i.e., the loss of architecture knowledge over time [1]. ADDs have been at the heart of software architecture knowledge management (SAKM) research, in which researchers are concerned with the capturing, sharing, using, and reuse of architecture knowledge [2].

Capturing architecture knowledge (i.e., documenting ADDs) and making it explicit can be regarded as prerequisite for subsequent SAKM activities, i.e., using, sharing, reusing, and maintaining architecture knowledge [3]. In the first 10 years of SAKM research, many approaches for capturing ADDs have been proposed. According to Capilla et al. [4], there still exist many barriers for the adoption of SAKM approaches in industry. Main barriers are the lack of adequate tool support, the high efforts required for capturing architecture knowledge, the disruption of the design flow caused by documenting architecture knowledge, and the lack of clarity about which architecture knowledge should be documented.

The associate editor coordinating the review of this manuscript and approving it for publication was Francisco J. Garcia-Penalvo.

To overcome these barriers, Capilla et al. propose to systematize the capturing process and to develop lean approaches for capturing to reduce the effort involved in capturing architecture knowledge.

In 2011, Nygard [5] proposed architecture decision records (ADRs) as a lightweight means for capturing ADDs. An ADR is a simple text file using a formatting language like Markdown that records a single ADD by using a simple template (similar to describing a pattern). ADRs are stored along other project artifacts in a repository that uses a version control system. ADRs seek to address some of the barriers mentioned by Capilla et al. [4]. They are a lightweight approach which requires only minimal resources and no dedicated tools except a simple text editor, and which systematizes the capturing process by means of a simple template. In 2018, ADRs were listed by the Thoughtworks Technology Radar as a technique that should be adopted [6].

Since ADRs have been suggested as means for capturing ADRs in practice, this raises the question if and to which degree ADRs have actually found their way into practice. In this paper, we conducted a mining software repositories (MSR) study to analyze the use of ADRs in open source repositories. We have systematically searched for repositories using ADRs at GitHub – the largest open source repository and development platform. The aim of our work is to investigate if and to which degree ADRs are used in practice and to get insights into current practices of using ADRs. Additionally, we identified future research challenges and discuss implications as well as recommendations based on the study's findings. To the best of our knowledge, this is the largest empirical study w.r.t. to the number of analyzed repositories.

This MSR study provides the following contributions:

1) We analyze the current adoption of ADRs by conducting the first comprehensive survey of existing ADRs across all available open source GitHub repositories.
2) We analyze how ADRs are actually used by practitioners in open source projects.
3) We provide a dataset, i.e., a collection of 921 GitHub repositories actually using ADRs, for creating a solid foundation to apply advanced analysis and learning techniques in order to support developers and software architects (e.g. in reusing, finding similar ADRs, etc.) in the field of SAKM and ADRs.

The remainder of this paper is organized as follows: In Section II, we provide a brief overview of the context of our work, i.e., ADDs and ADRs, and we survey related work. Section III describes our research approach, i.e., the research methodology we followed, as well as the planning of our study. In Section IV, we describe the execution phase of our study. We report our findings in Section V. In Section VI, we elaborate on implications as well as recommendations according to the study's findings and discuss the threats to validity of our research. Section VII concludes this paper with future work on further employment of our study results.

II. BACKGROUND AND RELATED WORK

In this section, we introduce the research background related to ADDs/ADRs and discuss related research.

A. BACKGROUND

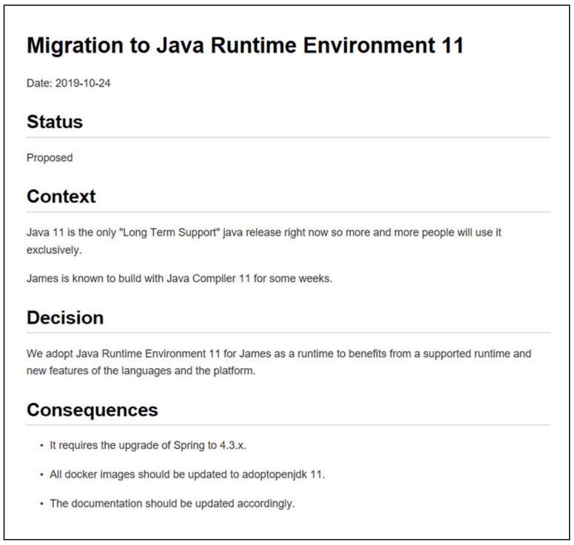
For a long time, software architecture was defined primarily as the structural organization of a software system in form of multiple structures (e.g., in [7]). Since 2004, the view on software architecture has been extended to also include the decisions that lead to the definition of various system structures as first class entities [1]. These decisions are called Architecture Design Decisions (ADD) [8]. The aim of ADDs is to capture the rationale behind a decision including design rules and constraints as well as additional requirements that have to be considered. Without ADDs essential design knowledge remains implicit in the heads of software architects and is subject of architecture knowledge vaporization [1], i.e., the loss of architecture knowledge over time.

In [9], Kruchten identifies three types of ADDs: *Existence decisions*, *property decisions*, and *executive decisions*. A typical example for an existence decision is the presence (or even absence) of a particular element or artifact in the resulting architecture, e.g., "The system is using PrimeNg as the sole library of basic UI components". Property decisions describe a lasting characteristic of the system, e.g., *"Each business function has to follow the MVVM pattern"*. Finally, executive decisions have an even more fundamental character and usually originate from the financial and/or organizational context of a software project, e.g., *"The system is developed with Angular/Typescript, starting with Angular version 14"* or *"Each inclusion of a dependency requiring a commercial license must be approved by the project steering committee"*.

After 2004, architecture knowledge and its management has become an active field of research [2], [4]. Farenhorst and Boer provide a classification for architecture knowledge in [10]. They distinguish between implicit (tacit) and explicit as well as between application-generic and application-specific architecture knowledge. Implicit knowledge refers to knowledge that only exists in the head of software architects and developers, while explicit knowledge refers to knowledge that has been made explicit by writing it down, e.g., in books, articles, magazines, and software documentation. Application-generic knowledge like architectural patterns, styles, and reference architectures represents knowledge that can be reused for the development of multiple applications, while application-specific knowledge like requirements and ADDs represents knowledge that is relevant for a specific application.

An ADR captures a single ADD and its rationale in a template-based document. Multiple templates for ADRs have been proposed by researchers as well as practitioners. The ADR templates encountered during our study are shown in Table 1. A sample[1] of an ADR using the template as proposed by Nygard [5] is depicted in Figure 1, showing the

[1]https://github.com/ASethi93/james.git (Accessed: May 24, 2023)



Migration to Java Runtime Environment 11

Date: 2019-10-24

Status

Proposed

Context

Java 11 is the only "Long Term Support" java release right now so more and more people will use it exclusively.

James is known to build with Java Compiler 11 for some weeks.

Decision

We adopt Java Runtime Environment 11 for James as a runtime to benefits from a supported runtime and new features of the languages and the platform.

Consequences

- It requires the upgrade of Spring to 4.3.x.
- All docker images should be updated to adoptopenjdk 11.
- The documentation should be updated accordingly.

FIGURE 1. Sample of an ADR using the template proposed by Nygard.

recommended sections, i.e., Status, Context, Decision and Consequences. A comparison of Nygard's and other ADR templates can be found in [11].

A central aspect of SAKM research has been the documentation of ADDs with appropriate methods and tools [12], [13]. Tang et al. [14] provide a comparison of early SAKM tools. Nat Pryce proposed the *ADR Tools* project[2] for working with ADRs. *ADR Tools* is a command-line tool for working with ADRs that supports users in adding ADRs and visualizing ADR dependencies. More recently, support for ADRs has been added to the Structurizr,[3] a diagrams-as-code tool, which also supports the generation of a decision log-based on ADRs as well as the visualization of dependencies between ADRs similar to the *ADR Tools*.

B. RELATED WORKS

There exists a significant body of work in research on documenting architectural design decisions [12]. Tofan et al. [19] analysed 144 publications in a systematic mapping study on the state of research on architectural decisions until 2012. They concluded that only a few of the analysed studies report the use of architectural decisions in industry. Alexeeva et al. [20] analyzed 96 publications from 2004 to 2015 and developed a taxonomy for the classification of existing documentation approaches for architectural decisions. Although ADRs were not explicitly identified, they mention template-based approaches as semi-formal approaches for capturing architectural design decisions. They also only found few works on industrial experience for documenting ADDs and mention the investigation of documenting architectural decisions in practice as a useful research contribution.

Anvaari et al. [21] present an exploratory study on architectural decision making practices in six Norwegian companies in the electricity industry in 2013. According to Anvaari et al. only "*some of the companies document important architectural decisions*", either by keeping "meeting minutes" or by using an internal wiki for documentation. They provide no information on the use of ADRs as means for capturing ADDs.

Also in 2013, Tofan et al. [22] present a survey with 43 architects from industry in which they analyzed real-world architectural decisions. According to Tofan et al., architectural decisions take an average of eight working days. They also state that most decisions are the result of group decision making processes including multiple stakeholders.

Weinreich and Groher [38] present an interview study with European and US software architects with more than 13 years of professional experience in 2016, where they also investigated how architects document architecture decisions and their rationale in practice. The results showed that architects in industry used various means for documenting architecture decisions: "*None of the interviewees reported using knowledge or decision management tools to manage project-related architectural knowledge. Instead, they mentioned text documents (64 percent), source code (60 percent), wikis (56 percent), project diaries and meeting minutes (40 percent), and issue management systems (20 percent)*" [38]. The use of ADRs has not been explicitly mentioned in the study.

Keeling and Runde [23] share experiences from using ADRs over two years for the development of microservices in the *IBM Watson Discovery Service* project. The team has recorded over 80 ADRs in two dozen repositories describing the *"birth, maturation, and death of services"*. They report that it took them a few months of training until everybody could write good ADRs. In the beginning, not all recorded ADRs were actual decisions (but rather guidelines) and not all decisions were architectural. ADRs were actually used to document decisions after they have been made as the result of preceding discussions. Reviews of ADRs were conducted using GitHub's pull request process in a similar way as code reviews. Further, they have established a dedicated repository for crosscutting ADRs. ADRs helped them to keep designs aligned in the light of staff turnover.

Industrial experience reports on the use of ADRs can further be found in grey literature. Harhio [12] conducted a multivocal literature review on documenting ADDs in 2022. Results showed that scientific literature was mainly concerned with the development of tools for documenting ADDs, while grey literature focused on ADRs.

Osl [24] presents lessons learned from using ADRs at *willhaben* – a virtual marketplace for goods in Austria. Osl reports that the ADR practices at *willhaben* differ from recommended practices in various points: (1) They decided not to store ADRs co-located with source code but in a central enterprise wiki to make ADRs also accessible to non-technical stakeholders, and (2) to use a customized template that has evolved over time. Osl has been using ADRs for over two years, in which more than 80 ADRs have been recorded. According to Osl, ADRs are well suited for

[2]https://github.com/npryce/adr-tools (Accessed: May 24, 2023)
[3]https://structurizr.com/ (Accessed: May 24, 2023)

TABLE 1. List of proposed ADR templates.

Template Name	Description
Template proposed by Jeff Tyree and Art Akerman [15]	First template proposed for capuring ADRs that tried to capture all essential information on decisions.
Template proposed by Michael Nygard [5]	ADR template with focus on simplicity.
Markdown Architectural Decision Records (MADR) proposed by Oliver Kopp et al. [16]	Markdown template to support the architecture-centric task of decision making with developer toolchains.
Template for Alexandrian Pattern proposed by Joel Parker Henderson[a]	Decision record template inspired by the Alexandrian pattern for documenting patterns.
Template using Planguage proposed by Joel Parker Henderson[b]	An ADR template using Planguage, an informal, structured, keyword-driven planning language initially developed by Tom Gilb [17] for the specification of requirements.
Template proposed by Paulo Merson[c]	Template that was inspired by the template proposed by Michael Nygard, but extended with an extra section to also document the rationale behind decisions.
Template proposed by Fabian Keller[d]	ADR template inspired by other ADR templates with the advice to customize the template according to your needs.
Y-Statements proposed by Uwe Zdun et al. [18]	A light template consisting of six sections that was inspired by Architecture Haikus.

[a]https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/templates/decision-record-template-for-alexandrian-pattern/index.md (Accessed: May 24, 2023)
[b]https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/templates/decision-record-template-using-planguage/index.md (Accessed: May 24, 2023)
[c]https://github.com/pmerson/ADR-template (Accessed: May 24, 2023)
[d]https://gist.github.com/FaKeller/2f9c63b6e1d436abb7358b68bf396f57 (Accessed: May 24, 2023)

structuring discussions and can be used for communicating a meeting agenda. However, he also states that ADRs are not a good discussion format and that he uses communication via slack for the discussion process of ADDs.

According to Blake [25], a handful of teams at Spotify are using ADRs to document decisions related to system design and engineering best practices. The benefits of using ADRs are perceived to bring new team members up to speed during onboarding, to support ownership handover processes as part of organizational changes, and to align on best practices across the company and to reduce multiple variants of solutions.

ADRs are also used by the mobile team at GitHub for documenting decisions affecting iOS and Android codebases. According to Perkins [26], *"ADRs are not the most common within open source codebases, but have gained more popularity since 2017"*. Like Keeling and Runde in [23], Perkins also sees ADRs as the result of (lengthy) discussion processes. ADRs allow informing team members how and why decisions were made in an asynchronous fashion, which saves direct communication between team members and supports the onboarding of new teammates. Recording ADRs is considered useful for reflection in the future to recall how and why decisions were made.

To sum up, existing industrial experience reports on the use of ADRs only cover single companies, which does not permit deriving general statements regarding the general adoption of ADRs in practice. Corresponding research that analyses a larger number of projects using ADRs is still missing. This study lays out the foundation for an analysis of the adoption of ADRs in open source projects and provides a first attempt of analyzing ADR practices.

III. RESEARCH APPROACH

In this section, we describe the research methodology we followed in our study, as well as how the study was planned.

A. RESEARCH METHODOLOGY

Vidoni [27] proposes a systematic process and guidelines for MSR studies, i.e., the process to analyze and cross-link data available in open source repositories to uncover interesting and actionable information about software systems [28]. The proposed process is the result of a systematic literature review (SLR), in which existing MSR papers have been analyzed to derive a systematic process. Vidoni compares the process of MSR with the process of SLRs, since both methods can be classified as secondary studies, i.e., studies in which results are aggregated by analyzing existing data (primary studies) to derive stronger forms of evidence about a particular phenomenon [29]. The MSR process and guidelines proposed by Vidoni are based on guidelines for conducting SLRs [30]. Although MSR and SLR studies both have conceptual similarities, there are also important differences: MSR studies differ from SLR studies in their searched sources (repositories vs. digital libraries), the analyzed elements (software projects, source code, issue reports, etc. vs. research papers) and the analysis process (a process that can only begin after data collection and that is at least partially automated vs. a typically manually conducted process).

According to Vidoni [27], the steps for conducting a MSR study are the following:

1) *Definition of Sources*: The selection of directories in which the search for repositories will be conducted.
2) *Search for Repositories*: The search for repositories that contain the data that will be analyzed.

3) *Data Extraction*: The extraction of data from the identified repositories that will then be used for analysis.
4) *Information Generation*: The generation of information regarding the research questions that is extracted from the data.

Further, the phases of an MSR study can be separated into a *planning phase*, an *execution phase*, and a *results phase* [27]. In the planning phase, the research objectives and the way how the study is executed are defined. The execution phase is comprised of searching for repositories in the defined sources and the extraction of data from the found repositories. Finally, in the results phase, the extracted data is summarized and analyzed, and the results, i.e., the answers to the research questions, are presented.

Following the proposed steps and phases for MSR studies, we describe the planning phase of our study in the remainder of this section. We describe the execution phase of our study (the search for repositories and the data extraction) in Section IV and the result phase in Section V.

B. STUDY PLANNING

Study planning is divided into the description of research objectives, research questions, and the source and search strategy used for identifying the relevant repositories.

1) RESEARCH OBJECTIVES

The main objective we are pursuing with our study is to find out how widespread the use of ADRs has become. Existing experience reports (see Section II-B) currently only deal with one or a few companies but do not provide any insights into the general adoption of ADRs. We decided to analyze the adoption of ADRs in open source repositories at GitHub, because GitHub is the largest hosting platform for open source repositories and thus provides a huge data basis for our analysis. A further objective is to find out how ADRs are actually used by practitioners.

2) RESEARCH QUESTIONS

To achieve our research objectives, we defined the following research questions (RQs):

RQ1: How has the adoption of ADRs evolved over time?
RQ2: What is the state of practice of using ADRs in open source repositories?

The aim of RQ1 is to investigate how the adoption of ADRs has evolved over time since the concept and corresponding ADR templates have been proposed. RQ2 then analyzes open source repositories that are actually using ADRs and seeks to put light on the actually followed practices of capturing ADRs.

To answer RQ1, we investigate the following subquestions:

RQ1.1: How many GitHub repositories are using ADRs?
RQ1.2: When did each repository start using ADRs?
RQ1.3: How has the number of recorded ADRs developed over time?

The aim of RQ1.1 is to identify all GitHub repositories in which ADRs have been recorded, which is the foundation for answering all other RQs. RQ1.2 investigates when users started using ADRs in their repositories and if the adoption w.r.t. the number of repositories using ADRs has increased over time. RQ1.3 complements RQ1.1 by analyzing the adoption of ADRs based on the number of recorded ADRs on a yearly basis.

To answer RQ2, we investigate the following subquestions:

RQ2.1: How many ADRs are recorded per repository?
RQ2.2: How many users of a repository are contributing to ADRs?
RQ2.3: Are ADRs maintained over time?
RQ2.4: Over what period of time are ADRs recorded?
RQ2.5: Which ADR templates are used in open source repositories?

ADRs seek to capture decisions that are architecturally significant, i.e., which have a wide impact on a system's architecture. As part of RQ2.1 we analyze how many ADRs are typically captured for a repository. RQ2.2 analyzes how many users participate in capturing ADRs for a repository. ADRs further need to be updated when architectural decisions are changed. In RQ2.3 we analyze if recorded ADRs are maintained over time. As part of RQ2.4 we analyze over which period of time ADRs are recorded. For capturing ADRs multiple templates have been proposed. In RQ2.5 we analyze which templates have actually found their way into practice.

3) SOURCE AND SEARCH STRATEGY

To reach our research objectives and to answer the derived RQs we observe the use of ADRs as means for capturing ADDs in open source repositories. GitHub was selected as data source since GitHub is actually the largest hosting platform for open source projects. We have searched all public repositories for ADRs. To develop a corresponding search strategy, we first conducted a set of pilot searches using the search API provided by GitHub. These pilot searches helped us to get familiar with the search API, to find out if there are any ADR results at all, how we can carry out a systematic search with the API, and to develop strategies to circumvent certain API limitations we have encountered as part of the pilot searches. Further, the pilot searches helped us to derive inclusion and exclusion criteria.

Searching for ADRs was not straightforward due to several limitations of the search API provided by GitHub. At GitHub, the search string must not contain wildcard characters, which prohibited searching for headings of Markdown ADR templates, e.g., '##Context', '##Decision', or '##Consequences'. GitHub also limits the number of returned search results to 1000, which prohibits searching all repositories with a single query. Further, also the length of the search string itself is limited to 256 characters and at most 5 AND, OR, or NOT operators. Finally, the number of queries within a certain period of time is also limited, which slows down the entire search process dramatically.

To compensate the limited number of returned search results, we decided to break down the search process, and to limit queries for ADR files to single users. To compensate the possibility of searching for headings of Markdown templates we decided to search for Markdown files containing the word *"decision"* and further search the obtained results for ADR files. The rationale behind this search strategy was that each prominent ADR template (see Section II-B) at least specifies one section heading using the word decision. Due to the large search base and the search API quota limit, we automated the search process whenever possible.

IV. STUDY EXECUTION

A. SEARCH PROCESS (DATA COLLECTION)

In this section, we explain the steps we followed to search for GitHub repositories using ADRs. The main steps of the resulting search process are depicted in Figure 2.

We first collected a list of all GitHub users that we derived from searching for all public repositories (Step 1). In total, we collected 26.372.973 GitHub users with public repositories.

As a second step, we searched if a user possessed repositories with Markdown files that contained the word '*decision*'. In total, there existed 282.789 users with at least one repository with a Markdown file containing the word '*decision*'.

The results we obtained in Step 2 contained many false positives, i.e., Markdown files containing the word *"decision"*, that are actually no ADRs, due to the rather generic search term. For instance, results contained Markdown files describing decision trees, licence agreements (e.g., the GNU GENERAL PUBLIC LICENSE Version 2), and GitHub pages, i.e., repositories that are used for hosting static websites based on Markdown files. Thus, to find the actual ADR files we conducted an automated search based on filename patterns (see Step 3). This means, we searched for filenames including the relative paths containing the following patterns: *arch, adr, design, decision*. As result we obtained 7331 users.

In Step 4, we manually screened the results of Step 3 to decide whether a result set of a single user contained ADR files or not. In cases where we were unsure, we looked at the search result directly on the GitHub website to make a decision. GitHub shows a preview of the search results and also allows individual files to be opened in the browser. At the end of the screening process we identified 1058 users with at least one repository containing ADR files.

In the final step of the search process, we manually verified each search result. In this step, we divided the search results per user into individual repositories (in the case that a user possessed multiple repositories), applied the inclusion and exclusion criteria, and removed duplicated repositories from the result set. The inclusion and exclusion criteria are summarized in Table 2. The main inclusion criteria was to include all repositories that are actually using ADRs for documenting ADDs. We excluded repositories that only explained the concept of ADRs, repositories that only contained an empty ADR template but no concrete ADRs, repositories that only contained a single ADR that states that ADRs should be captured for the project,[4] repositories that contain obvious test and example decisions, repositories containing ADRs not written in English, repositories that contained obvious classroom exercises, and repositories that contained decision logs rather than ADRs. We decided to exclude decision logs, i.e., collections of ADDs within a single file because it is a slightly different concept and we could not answer all of our RQs for decision logs. We used a plagiarism checker for detecting duplicated repositories, i.e., repositories with identical decisions. Repositories with identical ADRs are often the result when cloning a repository. In case we detected repositories with identical ADRs, we tried to identify the repository in which the ADRs had initially been defined (based on the git history) and removed all repositories that have been cloned and copied from the original repository. At the end of this manual verification and data cleaning process we identified a number of repositories containing ADRs that we used for answering the RQs. To facilitate further research, we provide a list with the identified GitHub repositories.[5]

TABLE 2. Inclusion and exclusion criteria.

Criteria	
Inclusion	• All repositories using ADRs for documenting ADDs
Exclusion	• Repositories that only explain the concept of ADRs • Repositories that only contain an empty ADR template, but no actual ADRs • Repositories that only contain a single ADR that states that ADRs are recorded for the project, but no additional ADRs • Repositories that only contain obvious test and demo decisions • Repositories with ADRs not written in English • Repositories using architecture decision logs • Repositories in which ADRs have been defined as part of classroom exercises • Cloned and copied repositories that contain the same ADRs as the source repository

B. DATA EXTRACTION AND ANALYSIS

For analyzing the identified repositories and answering our research questions, we developed a tool for automating the data extraction process as much as possible. The tool consists of two main components: a crawler and an analyzer. The crawler receives a list of repositories as input. Each repository is defined by the remote URL and a list of

[4]This is a default decision that is automatically created when using the *ADR Tools* (see also Section II-A).

[5]https://github.com/software-competence-center-hagenberg/ADR-Study-Dataset

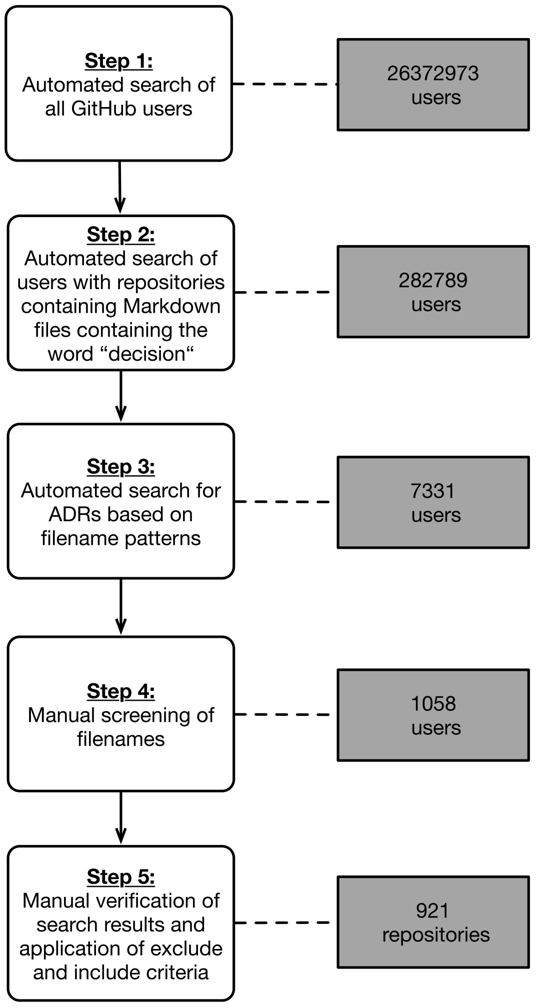

Step 1:
Automated search of all GitHub users
26372973 users
Step 2:
Automated search of users with repositories containing Markdown files containing the word "decision"
282789 users
Step 3:
Automated search for ADRs based on filename patterns
7331 users
Step 4:
Manual screening of filenames
1058 users
Step 5:
Manual verification of search results and application of exclude and include criteria
921 repositories

FIGURE 2. **Search process overview.**

directories, which contain ADR files. This list is derived from the semi-automated search process described above. The crawler downloads (clones) the repository and extracts the given directories for further analysis. The analyzer then reads and parses each file with the file extension *md* and collects metadata required for answering the research questions. For example, the analyzer counts the number of ADRs in each repository and tries to identify which ADR template has been used based on the headings in a Markdown file. If a file contains all headings of a template, it is classified as that template. If this is not possible because, e.g., some headings are missing or a custom template is used, the file is marked for a manual classification. Further, metadata is collected by using the Git history for each file (first commit date, last commit date, number of commits, and number of authors) and for the whole repository (initial commit date, last commit date). The tool was specifically tailored to collect the data necessary to answer the research questions.

After the initial crawl and analysis process, we checked each repository and the extracted ADR directories manually

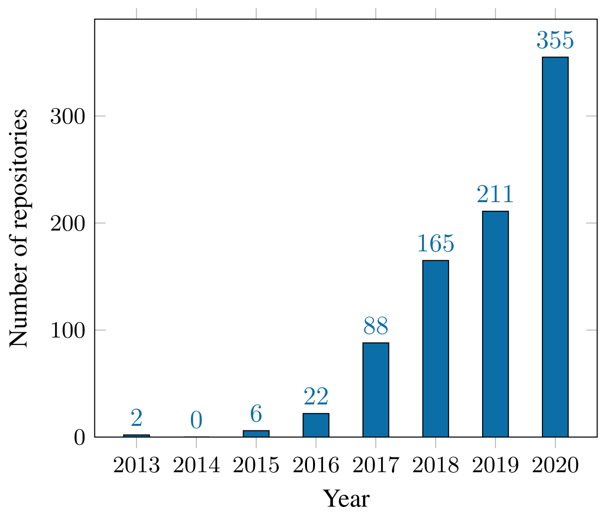


FIGURE 3. **Number of repositories starting to use ADRs per year. (RQ 1.2).**

to verify that our results only contained actual ADR files. All other files were removed (e.g., README files collocated with ADR files) and the collected metadata was updated to reflect the removed files.

1) HOW MANY GitHub REPOSITORIES ARE USING ADRs? (RQ1.1)

As result of our search process (see Section IV-A), we identified a total number of 921 GitHub repositories, in which at least one ADD has been captured using ADRs.

2) WHEN DID EACH REPOSITORY START USING ADRs? (RQ1.2)

To answer this RQ, we have analyzed, when the first ADR file was added to each repository. Analysis results are depicted in Figure 3. As shown in the figure, the first repositories starting using ADRs in 2013. In 2014 no new repository started to use ADRs, but in 2015 6 more repositories were found. From 2016 on there was a constant growth in repositories adopting ADRs, 22 new repositories in 2016, 88 in 2017, 165 in 2018, and 211 in 2019. Finally, in 2020 the most repositories so far started to use ADRs, specifically more than 350 repositories. The number of repositories depicted in Figure 3 is lower than the number of repositories identified as part of RQ1.1 – this is due to the fact that we do not have complete numbers for the year 2021 at the time of writing and thus could not include this data in the figure.

3) HOW DID THE NUMBER OF RECORDED ADRs DEVELOP OVER TIME? (RQ1.3)

In addition to analyzing when repositories started using ADRs (RQ1.2), we analyzed how many ADRs were added to GitHub repositories each year. Figure 4 provides an overview of the number of ADRs that were added to public GitHub repositories per year. As shown in the figure, the first ADR files were added in 2013 (2 ADRs), followed by 5 ADRs in 2014, 28 ADRs in 2015, and 118 ADRs in 2016. 2017 was

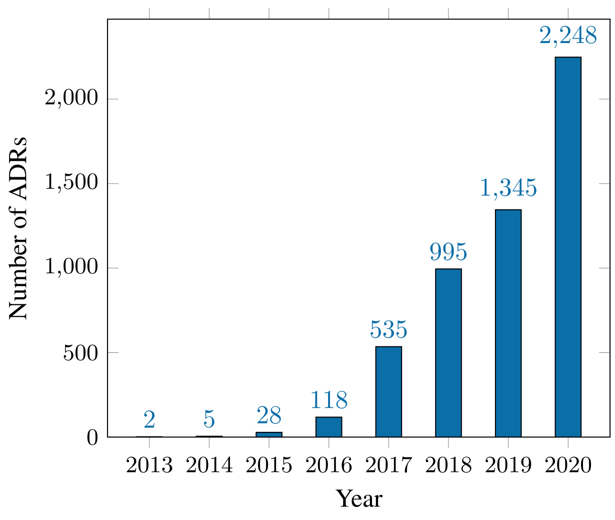


FIGURE 4. **Number of ADRs added per year. (RQ1.3).**

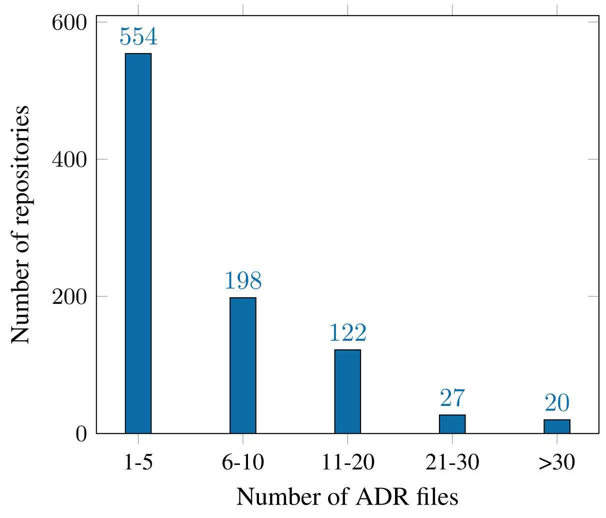


FIGURE 5. **Number of ADRs recorded per repository. (RQ2.1).**

the first year in which over 500 ADRs were added (i.e., 535 ADRs), followed by nearly 1000 ADRs (i.e., 995 ADRs in 2018). In 2019, 1345 ADRs have been added. The most ADRs (i.e., 2248 ADRs) have been added in 2020.

4) HOW MANY ADRs ARE RECORDED PER REPOSITORY? (RQ2.1)

To determine how many ADRs were recorded per repository (project), we counted the number of ADR files of each repository. Figure 5 shows our analysis results. More than 50% of all repositories (554) contain between 1 and 5 ADR files. 198 repositories contain between 6 and 10 ADRs, 122 repositories contain between 11 and 20 ADRs, 27 repositories contain between 21 and 30 ADRs, and 20 repositories contain more than 30 ADR files. The highest number of ADRs added to a repository was 73.[6]

[6]https://github.com/vwt-digital/operational-data-hub (Last Accessed: May 24, 2023)

5) HOW MANY USERS OF A REPOSITORY ARE CONTRIBUTING TO ADRs? (RQ2.2)

We counted the number of authors contributing to ADRs in each repository. The results are depicted in Figure 6 (left). Within all repositories with ADRs, there are 453 repositories where only a single user defined ADRs. This corresponds to nearly half of all repositories. There are significantly less repositories with two contributing authors (209 repositories). As further shown in the figure, there are also many repositories in which more than two authors contributed to ADRs. The repository with the highest number of contributors[7] had 60 contributors at the time of our analysis. We further analyzed the number of contributing authors per repository for repositories containing at least 10 ADRs. The results are shown in Figure 6 (right). As shown in the figure, when just considering the repositories with more than 10 ADRs, the number of repositories with only one ADR contributor (33) is significantly lower than the number of repositories with two or more contributors (165), whereas the number of repositories with one ADR contributor dominates (453) when looking at the complete set of repositories with ADRs (shown in the left diagram of Figure 6).

We further measured the commit quota of the most committing author of ADRs within a repository. This quota is the number of ADR related commits of the most committing author divided by the total number of ADR-related commits and was calculated for every repository. The results are shown in Figure 7. The distribution of quotas is grouped by the total number of ADR authors, e.g. the first boxplot in Figure 7 (left) shows the distribution for repositories with 2 ADR authors. This boxplot shows that in the case of two contributing authors, the more active author accounts for about 65% of all commits (median), while the values of the commit activity range between 50% and 97%. The distribution of the boxplots reveals that in general the commit activity is distributed among the contributing authors, although also repositories exist where single authors are contributing most ADR-related commits (e.g., see whiskers for repositories with 5 and 8 contributing authors in Figure 7). Considering only repositories with 10 or more ADRs, Figure 7 (right) shows a similar distribution, although the most active users typically have a higher commit quota.

6) ARE ADRs MAINTAINED OVER TIME? (RQ2.3)

For each ADR file we counted the number of commits to analyze how often ADRs are modified. Figure 8 (left) shows these results as a histogram. Most ADR files (3255) are created and only committed initially. After that they are never modified again. Significantly less files were modified at least once, e.g., 1497 files were committed two times 688 were committed three times. The highest number of commits on a single file is 58.[8]

[7]https://github.com/cosmos/cosmos-sdk (Accessed: May 24, 2023)

[8]https://github.com/otrv4/otrv4/blob/master/architecture-decisions/005-brace-keys.md (Accessed: May 24, 2023)

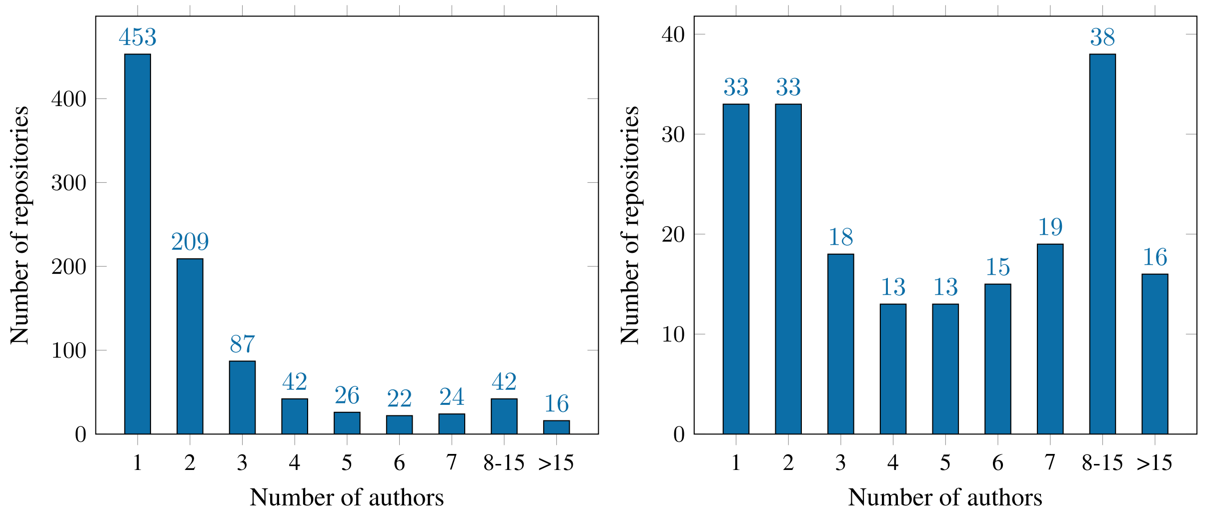


FIGURE 6. Number of repositories by number of contributing ADR authors. The left diagram shows the distribution of all repositories, the right diagram shows the distribution only of repositories with at least 10 ADRs. (RQ2.2).

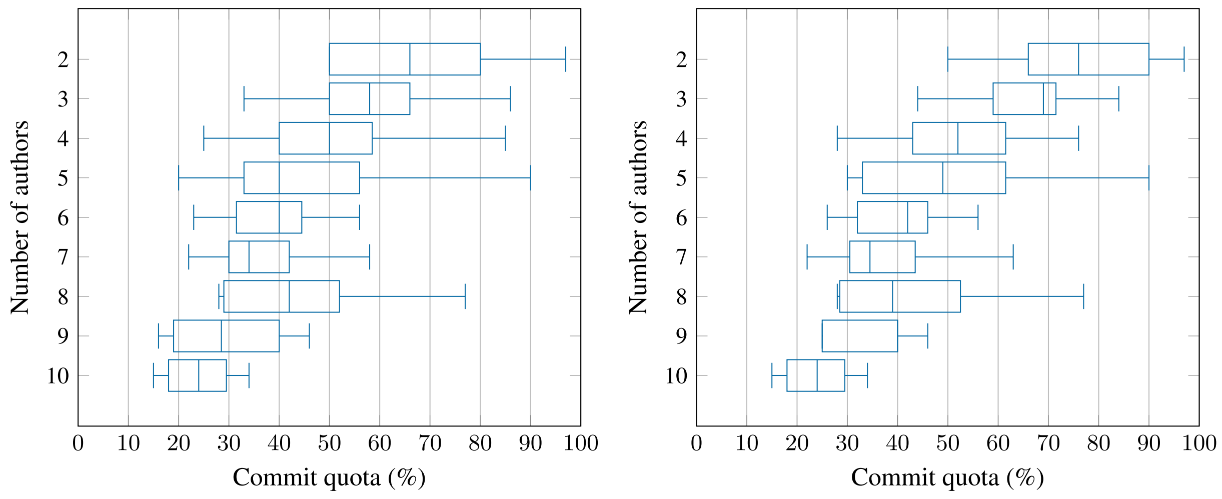


FIGURE 7. Commit quota of the most committing contributor. Only commits where an ADR file was affected are considered. The left plot shows the data of all repositories, the right plot shows the data only of repositories with at least 10 ADRs. (RQ2.2).

Figure 8 (right) shows the data only for ADR files within repositories with at least 10 ADRs. A similar distribution is visible here.

7) OVER WHAT PERIOD OF TIME ARE ADRs RECORDED? (RQ2.4)

For each repository, we analyzed the period over which ADRs were created and maintained by determining the initial commit date and the last commit date of each ADR file. Results are depicted in Figure 9 (left). In 278 repositories, all ADR files were only edited on a single day. This means, that they were created and potentially modified on the same day. Afterwards they were never touched again. In 100 repositories the editing timespan lasted between 2 and 14 days. In 38 repositories editing lasted between 15 days and one month. However, there also exist many repositories (505) with editing times between 1 month and more than two years, which means that in more than 50% of all repositories ADRs were maintained over a longer period of time.

Further, we analyzed the editing time period for repositories containing more than 10 ADRs. The results are shown in Figure 9 (right). Compared to considering all repositories, less ADR files are edited just for one day and more ADR files are edited for a longer period of time when focusing on repositories with more than 10 ADRs.

8) WHICH ADR TEMPLATES ARE USED IN OPEN SOURCE REPOSITORIES? (RQ2.5)

For capturing ADDs, multiple (lightweight) ADR templates have been proposed (see Section II). We analyzed which ADR templates are actually used. We identified the template used in each repository by matching the headings of each ADR file with the headings of each proposed template. In cases where no ADR template could be determined automatically, we manually inspected and classified the ADR files of the

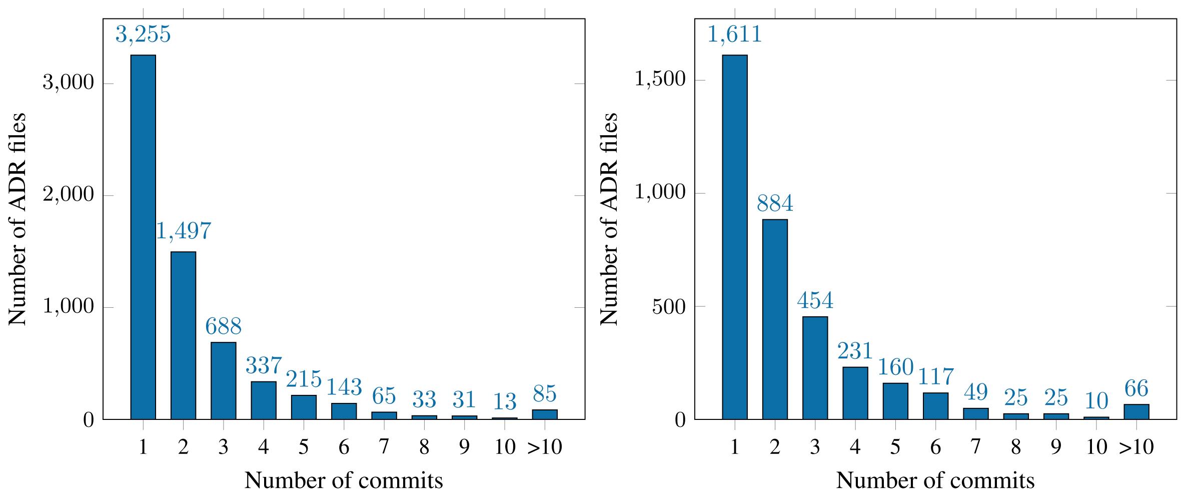


FIGURE 8. Number of commits per ADR file. The left diagram shows the distribution for all repositories, the right diagram shows the distribution only for repositories with at least 10 ADRs. (RQ2.3).

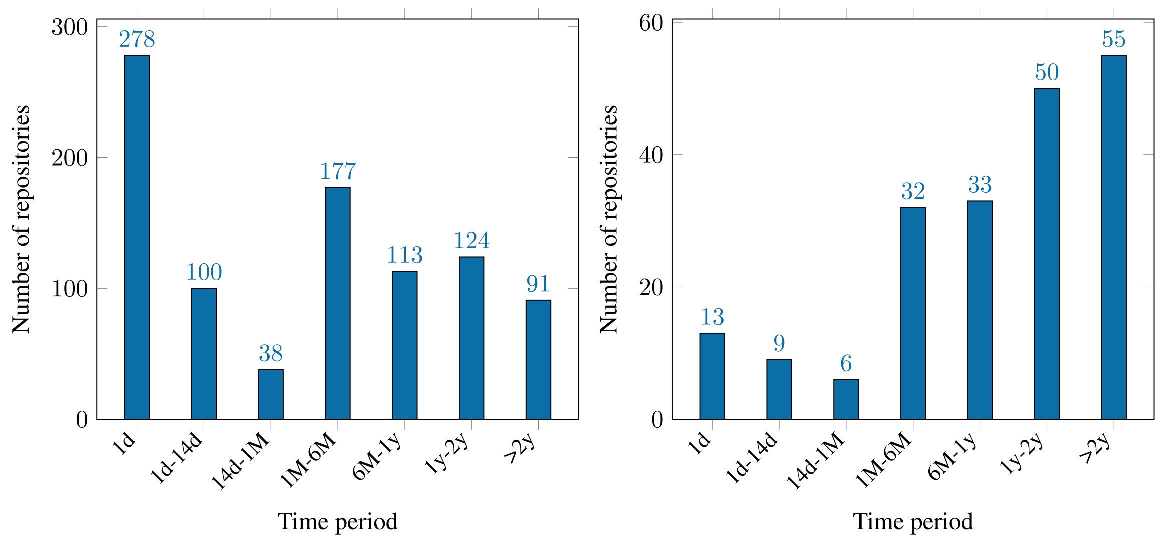


FIGURE 9. Number of repositories by the absolute editing time period, i.e. the time between the first and the last commit on an ADR file. The left diagram shows the distribution for all repositories, the right diagram shows the distribution only for repositories with at least 10 ADRs. (RQ2.4).

repository. The analysis results are shown in Figure 10. The template proposed by Michael Nygard is by far the most used template (723 repositories), followed by the MADR template (129 repositories). We also identified 61 repositories in which templates were used that did not correspond to known proposed/published templates. The other templates listed in Table 1 were only used by 8 repositories in total. The relative distribution of the templates being used does not change if only repositories with 10 or more ADRs are considered (see Figure 10 right). In this case the Michael Nygard's template is also dominant, followed by the MADR template.

V. STUDY RESULTS

In this section, we describe the results of our study. We answer the research questions based on the analysis results presented in Section IV-B and report our findings.

1) HOW MANY GitHub REPOSITORIES ARE USING ADRs? (RQ1.1)

We have found 921 public GitHub repositories that have at least recorded one ADD as ADR. This number does not reflect the number of repositories in which ADRs have systematically been recorded and used for a longer period of time, since this number also contains repositories in which only few ADRs (i.e., between 1-5 ADRs) have been captured. For many of these repositories, we assume that ADRs have been tried out, but the repository owners probably have decided not to further adopt ADRs. Nevertheless, for these found repositories we can conclude that the owner(s) of the repository are familiar with the concept of ADRs and that the repository owner(s) have at least tried out using ADRs for their projects. Given the number of public GitHub repositories (i.e., over 100 million public repositories), the number of repositories using ADRs is actually very small - especially

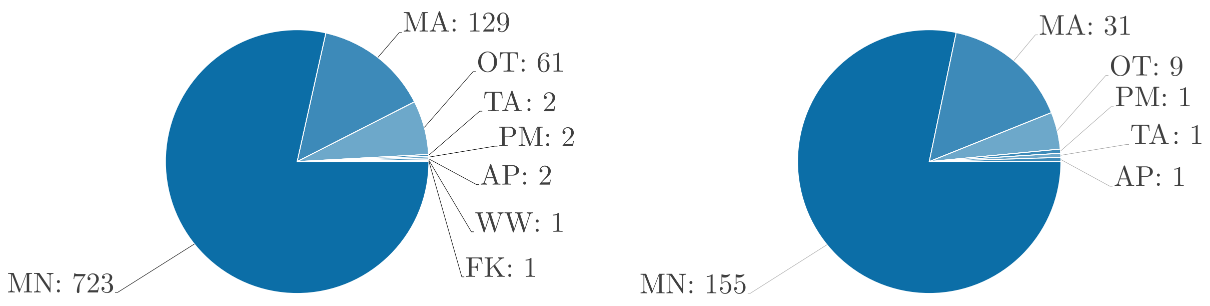


FIGURE 10. Number of repositories by the most used ADR template within a repository. The left chart visualizes the distribution for all repositories, the right chart shows the distribution only for repositories with at least 10 ADRs. Template abbreviations: MN…Template by Michael Nygard, MA…MADR, OT…Other, PM…Template by Paolo Merson, TA…Template by Jeff Tyree and Art Akerman, AP…Alexandrian Pattern, WW…Where What Why, FK…Template by Fabian Keller. (RQ2.5).

since not all repositories are using ADRs systematically (see also RQ2.1).

2) WHEN DID EACH REPOSITORY START USING ADRs? (RQ1.2)

The data on when individual repositories started using ADRs (depicted in Figure 3) shows that the adoption of ADRs started very slowly. Michael Nygard proposed ADRs in November 2011, the first GitHub repositories with ADRs date back to 2013 (2 repositories). By the end of 2015 only 8 repositories were using ADRs. Adoption started to increase in 2017, 5 years after the concept was introduced. Since 2017, we can observe an increasing adoption with yearly growth rates between 28% and 88%. Our data shows that the use of ADRs is still continuously increasing each year.

3) HOW DID THE NUMBER OF RECORDED ADRs DEVELOP OVER TIME? (RQ1.3)

Corresponding to the number of repositories that started using ADRs (RQ1.2), also the absolute number of recorded ADRs per year is increasing each year. Growth rates since 2017 range between 35% and 86%. Based on the analysis results of RQ1.2 and RQ1.3, we conclude that ADRs are seeing a steadily increasing adoption (within the analyzed time span), although at a low level when compared to the overall number of GitHub repositories. It remains to be seen how this adoption will develop in the following years.

Even if the number of projects using ADRs is still low, with over 2000 ADRs recorded in 2020 it would make sense – as part of future research – to systematically collect and analyze if and to which degree the recorded architectural knowledge (in terms of ADRs) is reusable for other projects.

4) HOW MANY ADRs ARE RECORDED PER REPOSITORY? (RQ2.1)

The number of ADRs recorded per repository differs significantly between different repositories. Thus, it does not make sense to calculate the average number of ADRs per repository and to derive statements regarding the typical number of captured ADRs in a repository or project. Nevertheless we can derive some findings from our analysis.

In more than 50% of all analyzed repositories, the number of added ADRs was just between 1 and 5 decisions. There can be different reasons for this low number of ADRs. It is likely that in these repositories capturing ADD using ADRs has been tried out, but has not been adopted systematically over time. It is natural that people try out an approach and stop using it if they are not fully convinced and do not see benefits for a further adoption. It would be a future research challenge to investigate the actual reasons for not adopting ADRs after trying them out. Conversely, a low number of ADRs does not automatically mean that ADRs were not fully recorded. Not every repository corresponds to one (large) project. It is also possible that a project comprises several repositories and the ADRs of a project are therefore distributed over several repositories, each with a few ADRs.

Statements about the number of ADDs in a typical project cannot be made by a quantitative analysis of the ADRs alone, but also require a corresponding qualitative analysis - either by reaching out to the repository owners, or by an in-depth analysis of each repository and the corresponding ADRs. A quantitative analysis of the identified repositories is beyond the scope of this study and remains subject of future research. However, with an increasing number of ADRs per repository, it is more likely that ADDs are captured systematically and that ADRs are successfully adopted in these repositories. For repositories with more than 20 captured ADRs, we conclude that there is a high probability that ADRs are adopted successfully in these repositories. However, from the total number of found repositories only 47 repositories (5%) contain more than 20 ADRs.

5) HOW MANY USERS OF A REPOSITORY ARE CONTRIBUTING TO ADRs? (RQ2.2)

In about 50% of the analyzed repositories, all ADRs were added by a single contributor only. This also means that in about 50% of all repositories ADRs were actually defined by more than one contributor and therefore decision recording was a team effort. In 130 repositories, five or more users contributed ADRs. If we restrict our analysis to repositories with at least 10 recorded ADRs (to exclude repositories in which ADRs probably only have been tried out but not been further adopted), it can be observed that in only 16% (33 of 198) of the repositories ADRs were defined by a single user, whereas in 84% of the repositories (165 of 198) multiple users made contributions to ADRs. We conclude that in projects where ADRs are adopted, decision making/recording is performed by the team and not by a single person. This finding is consistent with the findings of other researchers (e.g, [31], [38]) that architecting in practice is a group effort. This finding is also supported by the analysis of commit activity of the users contributing to ADR files.

6) ARE ADRs MAINTAINED OVER TIME? (RQ2.3)

In general, ADRs are not subject to frequent modifications. ADRs are intended to be collected following an append-only approach, where new decisions are added and obsolete decisions are marked with a superseeded status but kept for documentation purposes to show the history of the decision making [32]. Nevertheless, there can exist multiple reasons for making changes to an ADR, e.g., to simply correct a type, to change the status of an ADD, or to incrementally document an ADR over an extended period of the decision-making process.

Of the 6362 ADR files, about 50% of these ADRs have only been committed once and never modified again. This reflects that ADRs are not intended to be modified frequently. Regarding ADRs that have been modified at least once, analyzing the commit activity alone is insufficient to make statements regarding the performed modifications. This would require a more detailed analysis of the modifications made to ADR files, which is beyond the scope of our study and thus subject of future research.

7) OVER WHAT PERIOD OF TIME ARE ADRs RECORDED? (RQ2.4)

The timespan over which ADRs are recorded differs between the analyzed repositories. Many repositories show all modifications to ADRs being made within a single day – however, these are mainly repositories with only a few ADRs contained. When considering repositories with at least ten ADRs, repositories in which ADRs are only modified over a period of one day are the minority, while in all other repositories decision recording spans over a longer period of time. Our data reveals that in 328 repositories decisions are recorded over a period of more than six months. If we only consider repositories with at least ten decisions, 138 repositories have recorded ADRs over a period of over 6 months. Based on this data, we assume that in projects where ADRs are collected systematically, decision making/recording is a continuous activity that can span over multiple months and years.

A further reason that repositories only contain ADRs that have been modified over a timespan of a single day could be that ADRs have been recorded elsewhere and have been commited later to GitHub within a single commit operation.

A more detailed analysis, e.g., if there are times when most ADDs are recorded remains subject of future work.

8) WHICH ADR TEMPLATES ARE USED IN OPEN SOURCE REPOSITORIES? (RQ2.5)

Regarding the use of proposed ADR templates, it can be observed that the template proposed by Michael Nygard is the dominating template. It is being used in about 75% of all repositories. Also the MADR template is seeing adoption to a certain degree, followed by user-defined templates, while other templates do not play an important role. We could not observe significant differences regarding the selection of an ADR template between repositories with only few ADRs and repositories containing ten or more ADRs.

One possible reason that most repositories are using the template by Michael Nygard could be the fact that this template is used as default template by the *ADR Tools*. Thus, every repository using this tool for working with ADRs starts using automatically the template by Michael Nygard.

Regarding the use of custom templates, we observed that at least in some cases the use of a customized template was a dedicated decision because the users were not satisfied with existing templates. For example, in one repository[9] the MADR template was criticized for its verbosity. Another repository[10]) recorded the selection of a corresponding template as an own ADD in which different templates were listed as decision alternatives.

Our analysis also shows that complex ADR templates with many sections (e.g., the template proposed by Tyree and Akerman [15]) play no role for capturing ADDs in GitHub repositories.

A more detailed analysis of template usage, e.g., which sections of a template are actually used remains subject of future research. Such an analysis could provide insights, which aspects of ADDs are actually recorded in ADRs.

VI. DISCUSSION

ADRs were proposed as a lightweight approach (w.r.t. the demand of human resources) with the aim of making the capturing of ADDs practically feasible and to lay the foundation for the successful application of SAKM practices. With this goal in mind, we investigated whether ADRs are actually used in open source projects.

Based on our systematic analysis of public GitHub repositories we found that the adoption of ADRs in open source projects is still very low, although the adoption is steadily increasing in the analyzed time period. This finding is

[9]https://github.com/hugolhafner/analytics-server/blob/master/docs/adr/001-record-architectural-decisions.md (Accessed: May 24, 2023)

[10]https://github.com/croz-ltd/klokwrk-project/blob/master/support/documentation/adr/content/0001-architectural-decision-records.md (Accessed: May 24, 2023)

especially true since only a small number (i.e., 1-5) of ADRs were found in over 50% of all identified repositories and only 47 repositories contain more than 20 ADRs. For the majority of repositories with only few captured ADRs we assume that capturing ADD with ADRs has been tried without further adopting the practice.

The currently limited adoption of ADRs raises the questions why the adoption is still so low and how the adoption can be increased in the future? Finding answers to these questions requires further research that is not only comprised of quantitative analysis (as conducted in this MSR study) but that also encompasses qualitative analysis and that involves discussions with repository owners that decided not to adopt ADRs after trying them out.

However, we should also note that we have found a small number of repositories for which we conclude that ADRs have been adopted successfully. Figure 11 depicts the relationships between the number of ADRs per repository, the number of contributing users, and the time period over which ADRs are defined and modified. The VENN diagram visualizes repositories with at least 20 defined ADRs, repositories in which ADRs were edited for at least 6 months, and where at least two users contributed to the ADRs. As shown in the figure, there is a correlation between these aspects. 42 repositories can be characterized by these attributes, which hint at a systematic application and successful adoption of ADRs. For these repositories, we can conclude that writing ADRs (and probably also the preceding decision making processes) are typically conducted as a team process.

Multiple templates have been proposed for capturing ADDs. Given the dominance of the template proposed by Michael Nygard, we conclude that this template actually meets the needs of its users in most cases.

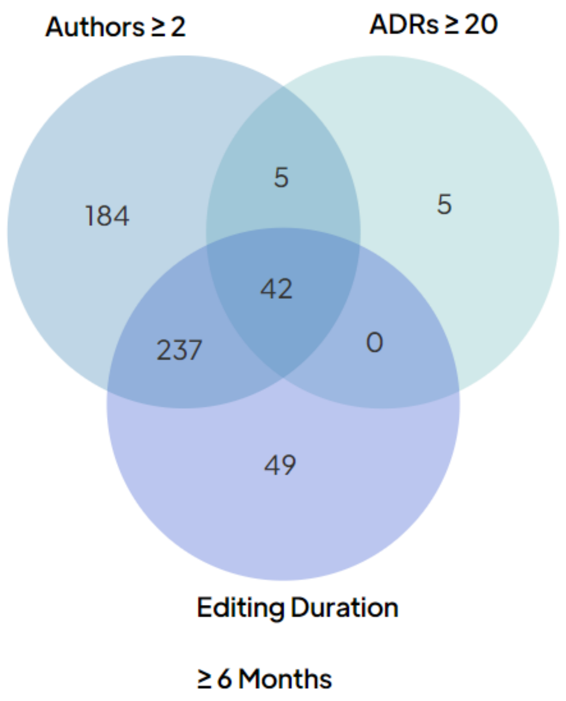


FIGURE 11. Overlap of repositories containing at least 20 ADRs that were contributed by at least two authors, and that have been added over a period of at least 6 months.

A. IMPLICATIONS OF STUDY RESULTS

Based on our findings, we have identified the following implications and recommendations for research and practice:

Simple organizational measures can enhance the discovery of ADRs in open repositories: ADRs not only make architecture knowledge explicit, but also facilitate architecture knowledge reuse across project boundaries. However, to support reuse ADDs/ADRs actually need to be discoverable. As we have described in Section IV-A, searching for ADRs is currently very tedious and resource-intensive w.r.t. human resources (required for manual search steps) as well as time resources due to API limitations. Therefore, a regular and automated search for ADRs at GitHub is currently not possible.

Recommendation: One possibility for improving the search for ADRs could be that ADRs use an explicit file extension (e.g., *.ADR instead of the standard Markdown file extension). This would facilitate explicitly searching for ADR files (using a dedicated file type extension filter) with currently provided search APIs capabilities provided by GitHub. Manual steps of the search process would become obsolete and the search process could be completely automated. This would make the search repeatable and would facilitate the development of new use cases like building up a knowledge base of ADRs.

Further qualitative analysis can help to increase the adoption of ADRs: While our study provides first insights in using ADRs in open source projects, many open questions remain that require further and more detailed analyses of current ADR practices. For instance, it would be interesting to know which kinds of ADDs (according to proposed classification schemata like [9] and [33]) are typically captured using ADRs in open source projects, and what were the reasons for many repository owners not to further use ADRs in their projects.

Recommendation: Conducting further in-depth analysis of ADR practices will deliver valuable findings needed to further develop the ADR practices to increase ADR adoption. Here it is not only important to find out the reasons for the limited adoption but also to advance existing concepts, methods and tools.

Reuse of ADDs can benefit from cross-cutting knowledge bases: Large code/project repositories such as GitHub facilitate building up knowledge bases. In the case of architecture knowledge management, a knowledge base of ADDs/ADRs consisting of architecture knowledge shared from ideally thousands of projects can facilitate large scale architecture knowledge reuse.

Recommendation: Proper knowledge bases should be developed which allow searching for design decisions typically made for particular kinds of systems like microservice architecture or AI-based systems, searching for ADDs related to particular implementation technologies and frameworks, and distilling ADDs related to specific software engineering tasks like testing or continuous integration and delivery. Providing such derived knowledge can assist inexperienced software architects as well as software architects new to a particular system in their decision making processes. The general idea and a prototype of such a system has been presented in [34]. Supporting such tasks based on GitHub would provide a global knowledge base where the above mentioned questions could be addressed for developers.

Knowledge bases will add value to learning from ADDs: Closely related to building up a knowledge base for ADDs/ADRs is the topic of exploiting artificial intelligence (AI)-based methods in this context. While GitHub code repositories have been used for the development (training) of language models that can be used for AI-based program generation (e.g., [35]), exploiting public code bases like GitHub for learning-based approaches using ADDs has not been subject of research.

Recommendation: Additional research needs to be performed to answer questions like how many ADRs are required for the training of high quality models, which architecture-related activities can be supported with AI-based methods, how ADRs written in informally-defined, natural language can be used by AI-based methods in contrast to formally defined source code with precise syntax and semantics, and how humans (software architects) and AI-based systems can collaborate during architecture design processes [36].

External validity of our findings cannot be claimed regarding the adoption of ADRs in closed source projects: The findings of our MSR study apply to open source projects only, which were subject of our analysis. However, this raises the question if and to which degree our findings also apply to closed source projects.

Recommendation: While there exist experience reports of using ADRs from single companies (see also Section II-B), studies analyzing the adoption of ADRs in multiple companies are still missing. Such studies are needed to analyze if reports from single companies can be generalized.

B. THREATS TO VALIDITY

Several threats possibly affecting the validity of our work do exist. According to Vidoni [27], threats in MSR studies may either arise from the selection of repositories or from the data extraction process. In this section we discuss the steps we took to mitigate potential threats and biases in our study, and which threats to validity still exist.

According to Kalliamvakou et al. [37], research results can be biased by threats that arise during the repository selection process, for instance, by selecting inactive repositories, repositories with a low activity, personal repositories (i.e., repositories not intended for collaboration with multiple users). Further, not all GitHub repositories are used for software development projects, but also for hosting a website, or for classroom/training exercises.

To mitigate threats during the repository selection process, we did not restrict the mining process for ADRs to a subset of repositories. The initial search for ADRs at GitHub was performed on all public repositories. Repositories that were used for hosting a website and for classroom/training exercises were defined as exclude criteria (see Section IV-A). These repositories were detected as part of a manual inspection step, in which each candidate repository was inspected by at least one of the authors of this study. For forked and cloned repositories, we tried to identify the repository in which ADRs were initially added. Cloned and forked repositories with identical ADRs as the base repository were defined as exclusion criteria.

Threats impacting the data extraction process are that not all activity on GitHub can be traced to registered users and that projects do not use GitHub exclusively [37]. The first threat cannot be mitigated because data provided by GitHub does not allow to identify activities of non-registered users. As a consequence it is possible that the number of committers to ADRs is higher as actually described as part of RQ2.2. We were also not able to mitigate the threat that repositories manage ADRs outside of GitHub (e.g., in a Wiki). Mitigating this threat is not possible, since this would require to investigate for each public repository hosted on GitHub if they maintain ADRs in some infrastructure outside of GitHub.

Finally, it needs to be taken into consideration that GitHub is continuously evolving [37] and repositories change on a daily basis. The conducted mining process described in this paper spanned over a period of multiple months – due to the API limitations provided by GitHub and the time-intensive work of manually analyzing the identified repositories. We weren't able to account for repositories where ADR files were added at a certain point in time but have since been deleted, because the GitHub search API doesn't allow us to search for deleted files.

To mitigate the threat of experimenter bias, multiple researchers (i.e., all authors) were involved in this MSR study. To ensure connstruct validity, the research questions were clearly defined and methods for collecting data to address the research questions were systematically selected.

Regarding external validity, we cannot draw general conclusions of ADR practices from open source projects to industrial closed source projects. Regarding external validity of our findings with regard to open source projects in general: With GitHub we decided to analyze the world's largest open source hosting platform and we have searched all public repositories that were available at the end of 2020. Thus we believe that the number of considered/searched repositories is large enough to draw conclusions (e.g., regarding the limited adoption of ADRs) from our results to open source projects in general.

VII. CONCLUSION

In this paper, we presented a MSR study on the adoption of ADRs in open source projects at GitHub. ADRs have been proposed as lightweight technique for the capturing of ADDs. This has raised the question if and to which degree ADRs have actually found their way into practice. Therefore, we developed a systematic search process to answer two main research questions regarding the adoption of ADRs over time as well as current ADR practices.

We found 921 repositories out of 1267 repositories in total, in which over 5000 ADRs have been defined. After a slow adoption process between 2013 and 2015, adoption started to increase steadily each year since 2017. Given the number of recorded ADRs per repository, we conclude that in many repositories the practice of recording ADDs with ADRs has been tried out but not adopted over a longer period of time. However, we also found repositories in which many ADRs have been recorded over a longer period of time. For these repositories, we conclude that ADRs have been successfully adopted. Regarding the number of contributors, in more than 50% of the repositories ADRs have been defined by more than one user, which hints that decision documentation (and thus presumably also decision making) is a team process. Regarding the use of proposed templates, we observed that the template proposed by Michael Nygard is the dominating template used by about 75% of all repositories. This shows that it seems to be sufficient for projects having adopted an ADR approach. However, it could also be an indication of the usefulness of tool support for capturing such decisions, since this template is supported by a dedicated tool. The MADR template is also seeing adoption to a certain degree, as well as templates that are customized to specific needs of their users.

Our study has shown that the adoption of ADRs as a well established practice is still in its infancy. Nonetheless, its prevalence is increasing year by year and it remains to be seen how the adoption of ADRs will develop over the next years. We also pointed out implications of the study's finding and give some recommendations to further facilitate the use of ARDs and to increase their adoption. Promising research directions are, e.g., the reuse of the recorded architecture knowledge on a global basis and the use of AI-based methods for identifying relevant knowledge. Finally, we employ our results in future research work, where we investigate the application of graph technologies for analyzing ADRs, thereby providing access to domain knowledge and enabling the discovery of additional insights. The core of our approach is the creation of a knowledge graph from ADRs, followed by the conversion of this data into Wikidata concepts. We believe that this approach could assist in the deeper analysis of current ADR practices and the creation of new ADRs by proposing best practices in specific domains.

REFERENCES

[1] J. Bosch, "Software architecture: The next step," in *Software Architecture* (Lecture Notes in Computer Science), F. Oquendo, B. C. Warboys, and R. Morrison, Eds. Berlin, Germany: Springer, 2004, pp. 194–199.
[2] M. A. Babar, T. Dingsøyr, P. Lago, and H. V. Vliet, *Software Architecture Knowledge Management*. Cham, Switzerland: Springer, 2009.
[3] R. Weinreich and I. Groher, "Software architecture knowledge management approaches and their support for knowledge management activities: A systematic literature review," *Inf. Softw. Technol.*, vol. 80, pp. 265–286, Dec. 2016.
[4] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, "10 years of software architecture knowledge management: Practice and future," *J. Syst. Softw.*, vol. 116, pp. 191–205, Jun. 2016.
[5] M. Nygard. (2011). *Documenting Architecture Decisions*. [Online]. Available: http://thinkrelevance.com/blog/2011/11/15/documentingarchitecture-decisions
[6] R. Parsons, M. Fowler, and B. Subramaniam. (2018). *Technology Radar*. [Online]. Available: https://www.thoughtworks.com/radar
[7] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Reading, MA, USA: Addison-Wesley, 2003.
[8] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Proc. 5th Work. IEEE/IFIP Conf. Softw. Archit.*, Oct. 2005, pp. 109–120.
[9] P. Kruchten, "An ontology of architectural design decisions in software intensive systems," in *Proc. 2nd Groningen Workshop Softw. Variability*, 2004, pp. 54–61.
[10] R. Farenhorst and R. C. D. Boer, "Knowledge management in software architecture: State of the art," in *Software Architecture Knowledge Management*. Berlin, Germany: Springer, 2009, p. 21.
[11] O. Zimmermann, L. Wegmann, H. Koziolek, and T. Goldschmidt, "Architectural decision guidance across projects–problem space modeling, decision backlog management and cloud computing knowledge," in *Proc. 12th Work. IEEE/IFIP Conf. Softw. Archit.*, May 2015, pp. 85–94.
[12] S. Harhio, "Documenting software architecture design decisions in continuous software development—A multivocal literature review," M.S. thesis, Fac. Sci., Univ. Helsinki, Helsinki, Finland, 2022.
[13] M. Keeling, "Love unrequited: The story of architecture, agile, and how architecture decision records brought them together," *IEEE Softw.*, vol. 39, no. 4, pp. 90–93, Jul. 2022.
[14] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. Ali Babar, "A comparative study of architecture knowledge management tools," *J. Syst. Softw.*, vol. 83, no. 3, pp. 352–370, Mar. 2010.
[15] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," *IEEE Softw.*, vol. 22, no. 2, pp. 19–27, Mar. 2005.
[16] O. Kopp, A. Armbruster, and O. Zimmermann, "Markdown architectural decision records: Format and tool support," in *Proc. ZEUS*, 2018, pp. 55–62.
[17] T. Gilb, *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Amsterdam, The Netherlands: Elsevier, 2005.
[18] U. Zdun, R. Capilla, H. Tran, and O. Zimmermann, "Sustainable architectural design decisions," *IEEE Softw.*, vol. 30, no. 6, pp. 46–53, Nov. 2013.
[19] D. Tofan, M. Galster, P. Avgeriou, and W. Schuitema, "Past and future of software architectural decisions—A systematic mapping study," *Inf. Softw. Technol.*, vol. 56, no. 8, pp. 850–872, Aug. 2014.
[20] Z. Alexeeva, D. Perez-Palacin, and R. Mirandola, "Design decision documentation: A literature overview," in *Proc. Eur. Conf. Softw. Archit.* Cham, Switzerland: Springer, 2016, pp. 84–101.
[21] M. Anvaari, R. Conradi, and L. Jaccheri, "Architectural decision-making in enterprises: Preliminary findings from an exploratory study in Norwegian electricity industry," in *Proc. Eur. Conf. Softw. Archit.* Cham, Switzerland: Springer, 2013, pp. 162–175.
[22] D. Tofan, M. Galster, and P. Avgeriou, "Difficulty of architectural decisions—A survey with professional architects," in *Proc. Eur. Conf. Softw. Archit.* Cham, Switzerland: Springer, 2013, pp. 192–199.
[23] M. Keeling and J. Runde. (Jul. 2018). *Distribute Design Authority With Architecture Decision Records*. [Online]. Available: https://www.agilealliance.org/resources/experience-reports/distributedesign-authority-with-architecture-decision-records/
[24] M. Osl. (Mar. 2021). *8 Learnings From Using Architecture Decision Records (ADRs) at Willhaben*. [Online]. Available: https://tech.willhaben.at/8-learnings-from-usingarchitecture-decision-records-adrs-at-willhaben-5b1594ebaffe

[25] J. Blake. (Apr. 2020). *When Should I Write an Architecture Decision Record*. [Online]. Available: https://engineering.atspotify.com/2020/04/14/when-should-iwrite-an-architecture-decision-record/

[26] E. Perkins. (Aug. 2020). *Why Write ADRs*. [Online]. Available: https://github.blog/2020-08-13-why-write-adrs/

[27] M. Vidoni, "A systematic process for mining software repositories: Results from a systematic literature review," *Inf. Softw. Technol.*, vol. 144, Apr. 2022, Art. no. 106791.

[28] A. E. Hassan, "The road ahead for mining software repositories," in *Proc. Frontiers Softw. Maintenance*, Sep. 2008, pp. 48–57.

[29] B. A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-Based Software Engineering and Systematic Reviews*. Boca Raton, FL, USA: CRC Press, 2015.

[30] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Inf. Softw. Technol.*, vol. 55, no. 12, pp. 2049–2075, Dec. 2013.

[31] H. Muccini, "Group decision-making in software architecture: A study on industrial practices," *Inf. Softw. Technol.*, vol. 101, pp. 51–63, Sep. 2018.

[32] M. Keeling, "The psychology of architecture decision records," *IEEE Softw.*, vol. 39, no. 6, pp. 114–117, Nov. 2022.

[33] C. Miesbauer and R. Weinreich, "Classification of design decisions—An expert survey in practice," in *Software Architecture* (Lecture Notes in Computer Science), K. Drira, Ed. Berlin, Germany: Springer, 2013, pp. 130–145.

[34] K. Brandner, B. Mayer, and R. Weinreich, "Software architecture knowledge sharing with the architecture knowledge base (AKB)," in *Proc. 13th Eur. Conf. Softw. Archit.*, New York, NY, USA, Sep. 2019, pp. 30–33.

[35] M. Chen, "Evaluating large language models trained on code," 2021, *arXiv:2107.03374*.

[36] M. Razavian, B. Paech, and A. Tang, "The vision of on-demand architectural knowledge systems as a decision-making companion," *J. Syst. Softw.*, vol. 198, Apr. 2023, Art. no. 111560.

[37] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining GitHub," *Empirical Softw. Eng.*, vol. 21, no. 5, pp. 2035–2071, Oct. 2016.

[38] R. Weinreich and I. Groher, "The architect's role in practice: From decision maker to knowledge manager?, *IEEE Softw.*, vol. 33, no. 6, pp. 63–69, 2016, doi: 10.1109/MS.2016.143.

GEORG BUCHGEHER received the Ph.D. degree in business and computer science from Johannes Kepler University Linz, in 2013. He has been a key Researcher and a Project Manager in the software science area with Software Competence Center Hagenberg GmbH (SCCH), since 2006. He is currently working as a Lead Software Architect with karriere.at GmbH. He has more than 16 years of experience in fundamental and applied research projects. He has published more than 45 scientific publications at international conferences and in scientific journals. His research interests include software architecture, software engineering, component- and service-based development, microservices, and knowledge graphs. Furthermore, he is a program committee member of multiple international conferences and workshops and serves as a reviewer for international journals.

STEFAN SCHÖBERL received the degree in software engineering from the University of Applied Sciences Upper Austria, Campus Hagenberg. Since 2021, he has been a Researcher and a Senior Software Engineer in the software science area with Software Competence Center Hagenberg GmbH (SCCH). He is also an External Lecturer with the University of Applied Sciences Upper Austria, Campus Hagenberg, since 2022. His research interests include compilers, software analysis, cloud computing, and software architecture.

VERENA GEIST received the Ph.D. degree in computer science from Johannes Kepler University Linz, in 2012. She is currently a key Researcher in the software science area with Software Competence Center Hagenberg GmbH (SCCH), where she leads the research focus on "complex systems analysis." She is a coauthor of numerous publications in scientific journals and conferences. She has more than 17 years of experience, as a researcher in the field of code analysis, software redocumentation, AI for software engineering, process-aware information systems, and knowledge graphs. Her signs of peer esteem are reflected in several invitations to program committees, reviewing journal submissions, and organization of workshops and conferences.

BERNHARD DORNINGER received the degree in business and computer science from Johannes Kepler University Linz. He has been a Senior Project Manager in the software science area with Software Competence Center Hagenberg GmbH (SCCH), since 2000. His research interests include requirements engineering, software analysis, and software architecture.

PHILIPP HAINDL received the degree in information and communication systems engineering and business informatics from the University of Applied Sciences Technikum Vienna and the Ph.D. degree in computer science from Johannes Kepler University Linz, in 2021. He is currently a Lecturer in software engineering with the St. Pölten University of Applied Sciences. He has more than 15 years of practical experience in industrial software projects as technology consultant, software engineer, and architect. His research interests include comprise software security, learning analytics in software engineering education, and resilient software architectures. He is a program committee member of international software engineering conferences and serves as a reviewer for international journals.

RAINER WEINREICH is currently an Associate Professor with the Department of Business Informatics–Software Engineering, Johannes Kepler University Linz (JKU). His research interests include software engineering and software architecture, with a current focus on software architecture knowledge management for large-scale enterprise software, cloud-native software architecture, and microservices. He is a member of program committees of leading conferences in software architecture and a regular reviewer for international conferences and scientific journals in software engineering. More information about him can be found at https://www.se.jku.at/weinreich.