2022 IEEE 19th International Conference on Software Architecture (ICSA) | 978-1-6654-1728-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSA53651.2022.00013

A Study on the Software Architecture Documentation Practices and Maturity in Open-Source Software Development

Michel Muszynski, Sven Lugtigheid, Fernando Castor, Sjaak Brinkkemper
Department of Information and Computing Sciences,
Utrecht University
Utrecht, The Netherlands
{m.muszynski, s.h.lugtigheid, f.j.castordelimafilho, s.brinkkemper}@uu.nl

***Abstract*—The best practices in the industry for Software Architecture (SA) documentation are not always followed, despite it being known that SA documentation can positively influence different aspects of software development. Open-Source Software (OSS) projects often operate in a different manner compared to proprietary software projects. This study investigates contemporary SA documentation practices in OSS projects to gain insights into: (1) what architectural elements are described, (2) what the different description formats and types are, and (3) what the maturity of the architecture description is. The SA description documentation of six OSS projects is identified, classified, and evaluated: VLC, OpenEHR, openKM, GIMP, Audacity, and Home Assistant. The results show that natural language is widely used in describing the architecture, sometimes accompanied by diagrams of informal models. The majority of documentation was found on websites and wikis. The maturity was evaluated by applying the Architectural Capability Model (ArchCaMo). Out of the five maturity levels, most projects did not get past the first level. Only one project reached the second level, and one project showed potential for level three as it was the only project with explicitly documented SA design decisions.**

***Index Terms*—Architecture description maturity, Open-source software, Software architecture, Software documentation**

I. Introduction

It has been over 20 years since the Open Source Initiative published *The Open Source Definition* [1], a document containing ten conditions a software license must adhere to in order to be certified as open source [2]. Since then, open-source software (OSS) has taken the world by storm. As of October 2021, the popular OSS platform SourceForge hosts over 500,000 projects[1]. This is indicative of a steady growth compared to 324,000 projects in March 2013 [3]. In addition, Github, a service that combines OSS project hosting and social network has more than 60 million registered users[2]. In July 2021, a UK-based survey among 273 businesses indicated that 97% of respondents used OSS, 89% ran OSS internally, and 65% contributed to OSS projects [4]. An analysis of 1,546 commercial codebases in 2020 found that 98% contained OSS components [5].

[1]https://sourceforge.net/about
[2]https://github.com/search?q=type:user&type=Users

The OSS community and the proprietary software industry have an entwined relationship of mutual influence. OSS has adopted practices from industry related to code reviewing, planning, and releasing [6]. Simultaneously, OSS has contributed to industry with development practices [7], [8] and technologies such as Git. The adoption of OSS development practices within industry has been referred to as "Corporate Open Source" [9] or "Inner Source" [10].

OSS usage is not without troubles though. The same analysis from 2020 [5] also revealed that 84% of the codebases had at least one vulnerability in an OSS component, with an average of 158 vulnerabilities per codebase [5]. Unbiased assessment of OSS quality is difficult to achieve [11]. The lack of consensus on objective OSS quality measures makes potential users rely more on personal perception. This introduces biases which have often affected OSS negatively [12]. Another factor inhibiting OSS adoption is the lack of accessible and up-to-date documentation which frustrates potential users that attempt to evaluate the software product [13]. An important type of software documentation that is directly related to software quality is software architecture (SA) documentation [14].

SA is the set of design decisions [15] and structures needed to reason about a system, which comprise software behaviour, software elements, relations among them, and properties of both [16]. SA allows communication with stakeholders through views which address certain concerns [17]. In 2014, a survey of 2000 OSS projects revealed that only 5.4% had any SA documentation [3]. The lack of documentation in OSS is similarly visible in requirements engineering. Conventional requirements analysis and specification are not necessary, and formal requirements specification documents are not typically mandated either in OSS [18]. The benefits of SA documentation have been claimed for years [19]–[22]. SA documentation is an educational tool to introduce new team members to a project [21]. In a scenario such as OSS development, where there are many individuals attempting to contribute for the first time [23], SA documentation could be beneficial to reduce the entry barrier. This is supported by Kazman et al. [24], who show that explicit SA documentation may indeed be beneficial to new contributors in OSS.

978-1-6654-1728-0/22/$31.00 ©2022 IEEE
DOI 10.1109/ICSA53651.2022.00013

This paper presents a study investigating the SA documentation practices in OSS. More specifically, we have analyzed the documentation of six mature, popular, and large open-source projects, identifying whether they have any SA documentation, the format of said documentation, and the different perspectives tackled by it, including the employed views. We have then evaluated the maturity of the SA documentation of these projects using the Architectural Capability Model (ArchCaMo) [25]. This model considers factors such as format and comprehensiveness of architecture descriptions, documentation of design decisions, and support for architecture improvement to assess the maturity of architecture documentation.

We found out that natural language is widely used in describing the architecture, sometimes accompanied by diagrams of informal models. The majority of the documentation was found on websites and wikis. Most projects have no means to record or manage architectural design decisions or the rationale associated to them, make limited use of formalised models, and do not explicitly account for stakeholders or viewpoints. Employed architectural views are typically ambiguous and semantically unclear. No project documents the constrains on the relationships between architectural components, nor track inconsistencies, or architecture evolution.

Three projects represent the two ends of the software architecture documentation spectrum among the analyzed projects: OpenEHR and Home Assistant, and GIMP. The former two are platforms that are expected to be extended in diverse ways. In this context, understanding the architecture of these platforms and the ways in which preexisting and new components interact is key. On the one hand, OpenEHR has extensive, semi-formal UML diagrams documenting multiple views, with an emphasis on the different stakeholders of the platform. It is the only project we examined with this level of documentation. On the other hand, it does not document its main design decisions or rationale behind them. Unlike OpenEHR, Home Assistant documents and manages decisions and rationale, but makes limited use of models and does not identify its stakeholders. Finally, even though the GIMP image manipulation program is the eldest of all the analyzed projects, it is the one with the least amount of architectural documentation; we could not identify models, documentation of design decisions or rationale, stakeholders, or direct references to the architecture of the system.

The remainder of the paper is organized as follows: Section II discusses the related work, Section III describes the study design and presents the formulated research questions, Section IV contains the results, Section V describes the overall findings with interpretations, Section VI addresses validity threats, and Section VII concludes with future research directions.

II. Related Work

The most comprehensive study on SA documentation practices in OSS was conducted by Ding and colleagues [3] who looked at 2000 OSS projects. They found that SA documentation has not been widely used in OSS: 108 projects had some SA documentation. Most of the found documentation was described informally through natural language (88.9%), diagrams were found in 41.7% of the projects, while no projects were found using formal models. Additionally, they found that most documentation was provided on a web-platform with HTML being used in 70.4% of projects. The lack of found SA documentation corroborates the issues raised by Levesque [26] that providing adequate and up-to-date documentation tends to be a problem in OSS projects. Interviews with OSS developers conducted by Michlmayr et al. [27] revealed that the lack of documentation is a common criticism and insufficient documentation may lead to communication issues and problems with attracting new contributors. A similar sentiment is portrayed in [28]. This survey among open-source python library contributors reported that respondents on average believed they should spend about 20% more time on documentation. Although some suggest documentation is a good place for new contributors to start, [29] identified a number of challenges: a lack of skill, technical barriers, a lack of standards, and a lack of incentive and credit. Code-level documentation was found to be present in some OSS projects, although its coverage and consistency varied greatly [30]. Documentation about higher level concerns is less frequent. The results from [31] show that UML diagrams are sparingly found in OSS development. That study described a GitHub mining approach using filters and heuristics in an automated process for the collection and identification of files containing UML. In the 1,240,000 analyzed repositories, 21,316 UML files were identified spread over 3295 GitHub projects (0.27%). The survey in [32] indicated that industry seems more fond of modelling. Out of 155 Italian software professionals, 63% indicated to use modelling at least sometimes.

The benefits of documentation and the drawbacks of its absence have also been studied. Ajila and Wu [33] conducted a survey questionnaire among 60 software intensive IT organizations investigating the effects of OSS adoption on development economics. They found that documentation was significantly positively correlated with budget, productivity, and quality. Documentation debt is incurred when software is modified but its corresponding documentation is not [34]. According to a survey and case study in [35], documentation debt can lead to adverse effects like: low maintainability, delivery delay, rework, and low external quality.

Architectural decay, also known as erosion or degradation [36], is defined as a gradually growing discrepancy between the planned and actual architecture where key quality attributes are no longer satisfied [37]. Le et al. [38] performed an empirical study of architectural decay in OSS. They analyzed 1263 recovered architectural views and found an average of 130 architectural smells per system version. The "smelly" parts of the system's architecture were statistically significantly more prone to changes and issues. The researchers concluded that architectural decay is undesirable and can be a source of problems for a software system. Multiple potential reasons have been identified for the occurrence of architecture decay in [36]. These include but are not limited

to: the rush of software evolution, recurring change, lack of architecture documentation, accumulating architectural debt, and accumulation of architectural design decisions.

This paper complements the current literature by studying SA documentation in large, popular open-source software systems. Unlike previous work with similar goals [3], it examines the kinds of architectural views that these projects employ. In addition, it evaluates the maturity of such documentation in terms of a maturity model. This is useful to highlight not only what elements are present in the architecture documentation but also what desirable elements are missing and thus in which ways the documentation can be improved.

III. Study Design

To investigate the architecture documentation practices in OSS projects, we conduct a study with large, mature open-source software projects. The unit of analysis of our study is the SA documentation of the OSS projects. The design of this study is elaborated as follows: Section III-A presents the research questions and their rationale, Section III-B introduces the target projects of this study, Section III-C describes the data collection process, Section III-D explains how SA elements are identified through an ISO standard and it explains how SA elements can be documented using different types of views, and Section III-E details the maturity model we employ to evaluate the SA documentation of the target projects.

A. Research Questions

The goal of this study is to gain insight into the current software architecture documentation practices in open-source software development. To accomplish this goal, the following main research question (MRQ) is formulated:

MRQ: *What are the software architecture documentation practices in open-source software development?*

The main research question can be decomposed into three separate research questions (RQs):

RQ1: *What parts of the software architecture are documented in open-source software?*

With the many different definitions of SA in literature, there is an equal lack of consensus on what SA description should contain [39], [40]. In order to answer this RQ, the architectural elements in the SA descriptions are identified by using the ISO/IEC/IEEE 42010:2011 standard "Systems and software engineering — Architecture description" [41] as a benchmark instrument. Additionally, the SA descriptions are classified according to three composed views. The intent of this RQ is: (1) to get an overview of which architectural elements are described in the documentation of OSS projects; (2) to get an overview of what broad views of the respective systems the documentation describes.

RQ2: *What are the documentation types used to describe software architecture in open-source software?*

While the ISO/IEC/IEEE 42010:2011 standard does describe what a SA description should contain, it explicitly states that it does not prescribe the process or method for the creation of SA description, nor does it specify any format or media for the recording of SA description [41]. The intent of this RQ is twofold: (1) to obtain an understanding of which documentation types and/or formats are used to document SA of OSS projects; (2) to gain insight into how formal the architectural descriptions are.

RQ3: *What is the maturity of the software architecture documentation in open-source software?*

There seems to be a lack of research in SA description maturity, let alone specifically in the context of OSS development. This RQ attempts to determine the maturity of the observed SA descriptions through the use of a proposed maturity model by Junior et al. [25]. Applying a maturity model allows for comparison with an established benchmark (i.e., the level requirements in the maturity model) [42] and it allows for comparison between OSS projects. The intent of this RQ is to conduct a structured evaluation of the SA descriptions by applying an architecture description maturity model in order to assess the current level of achievement and to identify areas for improvement.

B. Target Projects

The OSS target projects in this study were selected based on three main criteria: (1) the project is at least five years old; (2) the project operates in a different domain or provides distinctive functionalities compared to already selected projects; (3) the project's year of inception differs from already selected projects. The purpose of these criteria is to ensure that the selected cases are diverse and representative. The first criterion is to ensure that the projects have had time to establish themselves and to create a community. Practices may differ depending on operating domain or application type, hence the inclusion of criterion two. Similarly, there may exist a difference in practice depending on project age, leading to the formulation of criterion three.

Additionally, the representativeness is further enhanced by including projects which are business- or industry-focused, next to projects which target individual consumers. Lastly, a project was included which facilitates research and has strong ties with the academic world. The resulting selection of projects and some of their differentiating characteristics can be seen in Table I. In the remainder of this section we briefly describe each one.

VLC is a cross-platform multimedia player and framework that plays most multimedia files, various streaming protocols, and physical optical media (https://www.videolan.org/vlc/).

OpenEHR is an e-health platform consisting of open specifications, clinical models, and software which can be used to create standards, build information interoperability solutions, and facilitate research in healthcare (https://www.openehr.org/).

openKM is an Enterprise Content Management Software, often referred to as Document Management Systems (DMS).

It allows businesses to manage the production, storage, tracking, and distribution of electronic documents (https://www.openkm.com/).

GIMP is short for GNU Image Manipulation Program. It is a cross-platform image editor with a wide array of tools suitable for graphic designers, photographers, illustrators, and scientists (https://www.gimp.org/).

Audacity is a cross-platform multi-track audio editor and recorder. It supports recording, editing, effects, analysis, and custom plugins (https://www.audacityteam.org/).

Home Assistant is a home automation platform with a focus on local control and privacy. It provides support for many of the popular devices, services, and ecosystems in this domain (https://www.home-assistant.io/).

C. Data Collection

In this study, documentation containing SA description is identified and collected by manually scanning data sources. Third degree data collection is employed [43], i.e., independent analysis of SA artifacts that were already available. The scope of the data collection is limited by excluding the following documentation sources: source code, forums and/or message boards, issue trackers, code reviews systems, social media, and mailing lists. Although we recognize the relevance and importance of these data sources and the architectural information that they might contain, the volatility, lack of structure, as well as pragmatic considerations place them outside of the scope. Taking code review discussions as an example, these have been shown to contain design decision rationale [44]. However, a software project may have tens if not hundreds of thousands of such discussion messages, e.g., Home Assistant has over 34,000 closed pull requests[4], with a majority subject to code reviews. This makes their use as SA documentation impractical. For the selection of the documentation within scope, inclusion and exclusion criteria are defined.

Inclusion criteria:

IC1: The documentation is created within the OSS project.
IC2: The documentation contains at least one of the architecture elements described in the ISO/IEC/IEEE 42010:2011 standard.
IC3: The documentation can be interpreted without transformation and/or recovery, e.g., decisions documented in issue tracking systems or commit messages are not accounted for because they would need to be recovered before being used.

Exclusion criteria:

EC1: The documentation is not publicly available.
EC2: The documentation is not in English.

D. Identification and Classification of SA Elements

This section elaborates on the identification and classification of the SA elements found in the documentation. First, the high-level architecture description elements are identified by using the ISO/IEC/IEEE 42010:2011 standard. Second, a classification model is created based on the taxonomies of Kruchten [19] and Clements et al. [21], in order to get an overview of what parts of the system are described in the SA description.

1) Identification of architecture description elements: The ISO/IEC/IEEE 42010:2011 standard [41] is used to identify which architecture description element types are present in the SA description of the selected OSS projects. This standard prescribes that a SA description should contain the following elements:

Environment: The context determining the setting and circumstances of all influences upon a system.

System-of-interest: The system whose architecture is under consideration in the preparation of an architecture description.

Supplementary information: Information that has a support role in the documentation, e.g., date of issue, authors, reviewers, change history, scope, or glossary.

Stakeholders: All parties with interests in the system.

Concerns: The expressed interests of stakeholders in relation to the system.

Architecture viewpoints: A specification of the conventions for the construction, interpretation, and analysis of views which address concerns framed by the viewpoint.

Architecture views: A representation of the system which addresses concerns for stakeholders in accordance with the governing viewpoint.

Architecture models: A representation (often expressed in diagrams) of addressed concerns as specified by the viewpoint, grouped by the view.

Correspondences: The relationships between the various architecture description elements.

Correspondence rules: Constraints enforced on the correspondences.

Architecture decisions: A record of the key decisions made in the architecture design process.

Architecture rationales: A record of explanations, justifications, or reasoning about the architecture decisions that have been made [41].

2) Classification of SA in views: The classification model combines the views of the 4+1 architectural view model [19] and view types that are described by Clements et al. [21] into three views: functional view, run-time view, and deployment view. The 4+1 architectural view model considers four views: logical, development, process, and physical. Clements et al. categorizes SA elements in three distinct view types: module, component-and-connector, and allocation.

The logical view of [19] is about the functionalities that the system should provide. Additionally, it describes the abstraction levels between functionalities. The development view of [19] focuses on the software modules. The views describe their parts as modules. The logical view and development view of [19], and the module view type of [21] are combined into the "**functional view**" for the classification of SA elements.

[3]The OpenEHR project started with an exploratory phase in 1998. The development phase started in 2003 and transitioned into a community managed phase in 2014.

[4]https://github.com/home-assistant/core/pulls?page=2&q=is%3Apr+is%3Aclosed

TABLE I
PROPERTIES OF SELECTED OSS PROJECTS. THE NUMBERS WERE COLLECT ON NOVEMBER 10TH 2021.

OSS Project	Domain	Main language	Inception	Additional information
GIMP	Image editor	C	1995	47,525 commits, 100 contributors, and 3,400 stars on Github and Gitlab.
VLC	Mediaplayer	C	2001	90,355 commits, 554 contributors, and 8,200 stars on Github.
Audacity	Audio editor	C++	2000	14,141 commits, 146 contributors, and 6,600 stars on Github.
OpenEHR[3]	e-Health	Java	2003	OpenEHR is a bit different from the other projects in that it is actually a collection of standardization projects under the umbrella of an initiative involving 21 industry partners, 3 governmental organizations, and 11 professional members.
openKM	Document management	Java	2005	8,531 commits, 10 contributors, and 439 stars for the main project combining Github and SourceForge.
Home Assistant	Home automation	Python	2013	41,374 commits, 487 contributors, and 47,400 stars for the `core` project on Github.

The purpose of the functional view is to show the functionalities of a system statically. The functionalities can be shown at different abstraction levels and the functionalities can have relations with each other, and external projects. In our analysis of SA documentation, the projects can score four different values for the functional view: (1) mentions functionalities of different abstraction levels, (2) mentions only high-level functionalities, (3) mentions only low-level functionalities, (4) functionalities are not described. The labels that refer to these categories are: fully, high-level, low-level and none.

The process view of [19] describes a group of tasks that together form an executable unit. At the highest abstraction level, it only shows the sequence of the units that will be executed at run-time. At a lower level of abstraction, it additionally shows how the units communicate with each other. Component-connector patterns are used in this view as options for how the system communicates during run-time. The process view of [19] and the component-and-connector view type of [21] are combined in the **"run-time view"**. The run-time view is about showing how different elements in the system interact with each other during run-time. A project can score four different values for the run-time view: (1) mentions component-connector patterns and describes what elements interact with each other during run time, (2) mentions only component-connector patterns, (3) mentions only the elements that interact with each other during run time, (4) run-time elements are not described. The labels that refer to these categories are: fully, C-C patterns, interaction flow and none.

The third and final view is the **"deployment view"** which is based on the physical view of [19] and the allocation view type of [21]. The psychical view of [19] focuses on allocating software components on hardware components and is therefore part of the allocation view type. The deployment view is about what hardware is required for the software to work and how it communicates with the software. The projects can score three different values for the deployment view: (1) mentions the hardware components and describes the software of each component, (2) mentions the hardware component without the responding software, (3) hardware elements are not described. The labels that are refer to these categories are: fully, hardware and none.

E. Maturity Model

In order to assess the achievement of the contemporary SA documentation practices, a SA description maturity model is applied to the selected projects. The maturity model used in this study is the Architectural Capability Model (ArchCaMo) as proposed by [25]. ArchCaMo is a maturity model for SA description based on the ISO/IEC/IEEE 42010:2011. Table II shows the Key Architecture Items (KAIs) with their descriptions and the relevant sections of the ISO/IEC/IEEE standard.

ArchCaMo consists of five levels, with **level 1** being the basic starting level defined as: not having a formalized way to define the SA, some aspects deemed necessary for the project may be documented. **Level 2** (KAIs 2.1-2.8) describes the minimum architecture: an architecture description containing a set of architectural elements that an architecture description must contain, describing what the architecture entails. **Level 3** (KAIs 3.1-3.4) describes the defined architecture: an architecture description that not only describes the architectural elements, but also elaborates on the rationale behind their existence. **Level 4** (KAIs 4.1-4.3) describes the consistent architecture: an architecture description that enforces consistency among its architectural elements by employing frameworks and Architecture Description Languages (ADLs), and that keeps track of known inconsistencies. **level 5** (KAIs 5.1 & 5.2) describes the quantified and improved architecture, this meta-level aims to improve the architecting process by defining metrics on KAIs from previous levels and by establishing dedicated architecting responsibilities. An architecture description can only advance in maturity level if all KAIs from the previous level are satisfied, only satisfying some KAIs in a level is not enough to advance.

IV. RESULTS

The results are presented as follows: Section IV-A relates to RQ1 and describes which architectural elements were identified. Section IV-B is associated to RQ2 and elaborates on the employed document formats and the level for formality of SA descriptions. Section IV-C answers RQ3 and discusses the maturity of the SA description.

TABLE II
SA DESCRIPTION MATURITY MODEL: ARCHCAMO [25]

KAI	Description	ISO
2.1	Environment is identified	Item 4.2
2.2	Systems of Interest are identified	Item 5.2
2.3	Supplementary information is identified	Item 5.2
2.4	Stakeholders are identified	Item 5.3
2.5	Concerns for each Stakeholder are identified	Item 5.3
2.6	Viewpoints are defined	Item 5.4
2.7	Multiple Views are defined	Item 5.5
2.8	Models are defined	Item 5.6
3.1	Correspondence rules are identified	Item 5.7.2
3.2	Rationales for decisions are identified	Item 5.8.1
3.3	Concerns related to each decision are defined	Item 5.8.2
3.4	Decisions are managed	Item 5.8.2
4.1	Known inconsistencies are recorded	Item 5.7.1
4.2	Use of Architecture Frameworks	Item 6.1
4.3	Use of Architecture description languages	Item 6.3
5.1	Metrics on elements of the other levels are defined	Item 7
5.2	An Architecture Group is established	Page 2

A. Architecture Description Elements

The ISO/IEC/IEEE 42010:2011 standard suggests 12 architectural description elements that every SA description should contain [41], as elaborated in Section III-D1. The most common elements found are the system-of-interest and supplementary information, these were found in all projects. Views and models were found in all projects but GIMP. VLC and GIMP were the only projects not to describe environmental influences. Home Assistant is the only project that has explicitly documented architectural decisions with rationale. Stakeholders, concerns, and viewpoints were only documented by OpenEHR. Correspondences and correspondence rules were not found in any of the investigated projects.

In order to get an understanding of what parts of the system these projects document, three views have been composed based on [19] and [21], as described in Section III-D2. Table III shows that five out of six projects document their functionality. Audacity and VLC use Doxygen generated models to visualise their functionalities. OpenEHR has a lot of documentation with over 30 diagrams, in these diagrams they describe their functionalities and run-time elements. Half of the investigated projects describe either the component-connector patterns or the flow that is followed during run-time. Home Assistant and openKM are the only projects of the sample that describe the required hardware. This view is however not relevant for all projects, because some software products are meant to run on their users PC.

B. Architecture Description Format and Formality

1) Format: The selected projects tend to favor web-platforms to document their SA. VLC, openKM, and OpenEHR explicitly have SA description elements on their website. Wikis were also used by multiple projects, these being VLC, GIMP, and Audacity. OpenKM uses a custom hyperlink system for newer versions of their system while also keeping their deprecated wiki around for older versions. In addition to manually crafted documentation, VLC and Audacity use Doxygen[5] for tool generated documentation. This includes an overview of modules, classes, and even functions. Additionally, Doxygen also provides integration with a drawing toolkit to automatically generate diagrams and graphs, e.g, collaboration diagrams, inheritance diagrams, or caller graphs. Home Assistant keeps track of its architectural decisions in markdown files in their GitHub repository.

2) Formality: In previous work [3], SA descriptions in OSS documentation have been classified into three categories: natural language, informal models, and formal models. Natural language refers to informal textual descriptions. Informal models are defined as models without defined syntax or semantics, often visualized with a *lines-and-boxes* approach. Formal models refer to models that employ an ADL with formal syntax and semantics (e.g., ACME [45], Wright [46] or C2 SADL [47]). This study adds a fourth category: semi-formal models. These are models with a defined syntax, but which lack analysis capabilities that ADLs offer (e.g., UML). Table IV presents an overview of the documentation formality found in the selected OSS projects. If a project's SA documentation contained an element that was part of a category, the general presence of that category is denoted with a ✓.

The majority of the cases have one or multiple informal models which are often accompanied by natural language descriptions. The amount of informal models that the investigated projects have varies ranging from none to multiple dozens. OpenEHR uses multiple semi-formal UML models while VLC and Audacity make use of tool-generated semi-formal diagrams. Formal models were not found during this study.

C. Architecture Documentation Maturity

The maturity level is determined for the investigated projects according to the maturity model created by [25]. Table V specifies which projects satisfy which maturity requirement. The ✓ specifies that a project's SA description satisfies the required KAI.

The table shows that only OpenEHR managed to reach level 2 and the other projects do not get past the base level 1. In other words, they do not reach what this maturity model

[5]https://www.doxygen.nl/index.html

TABLE III
OVERVIEW OF VIEW INCLUSION IN OSS PROJECTS

OSS Project	Functional View	Run-time View	Deployment View
VLC	Fully	C-C patterns	None
OpenEHR	Fully	Fully	None
openKM	High-level	None	Hardware
GIMP	None	None	None
Audacity	Low-level	None	None
Home Assistant	High-level	Interaction-flow	Hardware

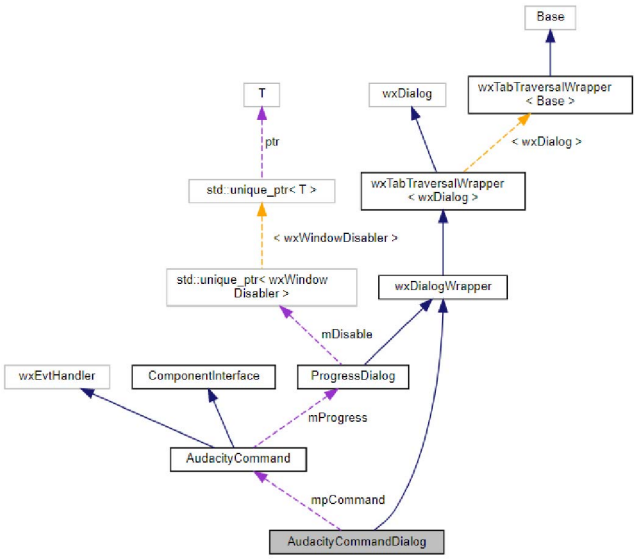

Fig. 1. Collaboration diagram of `AudacityCommandDialog` class

considers to be the minimum architecture documentation. Most of the projects do not satisfy KAIs 2.4-2.6. To satisfy these maturity requirements, projects need to specify governing viewpoints, the relevant concerns, and to which stakeholders these concerns belong. A different notable observation is that Home Assistant is the only OSS project to explicitly document SA decisions with rationale, managing to satisfy certain level 3 KAIs. Whether this is related to the fact that Home Assistant is the youngest project in this comparison, could be a direction for future research. Somewhat ironically, GIMP, the oldest among the analyzed projects, is the least mature project in terms of architecture documentation. We could not identify models, documentation of design decisions or rationale, stakeholders, or, more generally, direct references to the architecture of the system. None of the projects managed to satisfy any of the KAIs in level 4 or level 5. Overall the assessed SA descriptions are of low maturity: five level 1 maturities and one level 2 maturity.

V. Discussion

This section discusses the results, provides examples, and presents some other observations.

TABLE IV
SA description formality in OSS projects

OSS Project	Natural Language	Informal Models	Semi-Formal Models
VLC	✓	✓	✓
OpenEHR	✓	✓	✓
openKM	✓	✓	
GIMP	✓		
Audacity	✓	✓	✓
Home Assistant	✓	✓	

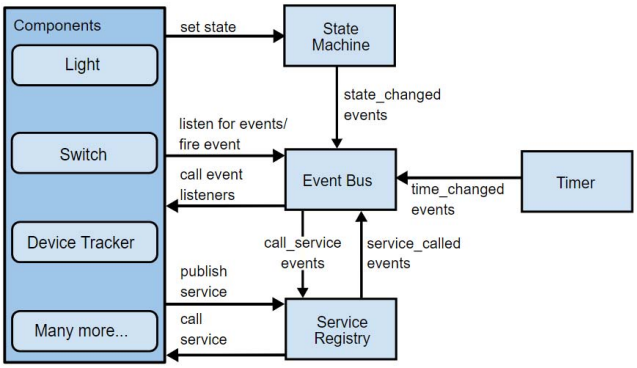

Fig. 2. Core Architecture of Home Assistant

The results show that all investigated projects document some aspect related to SA. There was no project that did not have any SA documentation. Most of the documentation can be found in web-based platforms with HTML being a popular format. Wikis are used by half of the projects, these provide a convenient way of creating, editing, and publishing documentation. The open, community-fuelled collaborative editing aligns well with the nature of OSS. Previous work in [48] has identified potential problems with wikis though: spam and URLs can bloat a wiki with content that is of little value, they lack authoritativeness, and their low barrier of entry may cause a decline of conciseness and focus. The only file-based approach that was observed is the SA decisions recording done by Home Assistant. They use markdown files in a separate part of their GitHub repository dedicated to decision recording. These files contain textual descriptions of the context, decision, and consequences. Each decision is documented in a separate file.

The wikis of VLC, GIMP, and Audacity provide user guides or manuals. VLC's wiki provides a "Hacker Guide" with numerous helpful articles to developers. Similarly, GIMP's wiki contains a "Hacking" category. Audacity also has development tutorials and how-to's in their wiki and Home Assistant makes extensive use of code fragments to support developers. One can infer implicit stakeholders from these observations, e.g, users and developers, yet no comprehensive overview of all stakeholders was found in most projects.

Most of the documents found are of informal nature. The majority of the projects do not have viewpoints defined. This observation is in line with the results from [3] who found in their survey that no OSS projects out of 108 documented viewpoints. The lack of well-defined viewpoints is not surprising since most projects also do not explicitly describe stakeholders and concerns. In addition, no project documents the constrains on the relationships between architectural components, nor tracks inconsistencies or architecture evolution.

A notable finding was the availability of tool generated documentation which includes semi-formal diagrams. VLC and Audacity provide Doxygen generated documentation in HTML format. Figure 1 shows an example of a semi-formal

TABLE V
SA DESCRIPTION MATURITY LEVELS OF THE SELECTED OSS PROJECTS

KAI	Description	VLC	OpenEHR	openKM	GIMP	Audacity	Home Assistant
2.1	Environment is identified		✓	✓		✓	✓
2.2	Systems of Interest are identified	✓	✓	✓	✓	✓	✓
2.3	Supplementary information is identified	✓	✓	✓	✓	✓	✓
2.4	Stakeholders are identified		✓				
2.5	Concerns for each Stakeholder are identified		✓				
2.6	Viewpoints are defined		✓				
2.7	Multiple Views are defined	✓	✓	✓		✓	✓
2.8	Models are defined	✓	✓	✓		✓	✓
3.1	Correspondence rules are identified						
3.2	Rationales for decisions are identified						✓
3.3	Concerns related to each decision are defined						✓
3.4	Decisions are managed						✓
4.1	Known inconsistencies are recorded						
4.2	Use of Architecture Frameworks						
4.3	Use of Architecture description languages						
5.1	Metrics on elements of the other levels are defined						
5.2	An Architecture Group is established						

collaboration diagram for the `AudacityCommandDialog` class[6], the diagram is accompanied by a legend[7]. The legend provides a clearly defined notation with different rectangles indicating whether the class is documented and different colored arrows indicating access modifiers, usage by other classes, and template instance relationships.

The general informality is sustained in the modelling and diagramming efforts of the investigated projects. Figure 2 is from the Home Assistant website[8] and shows what they call the *"core architecture"*. The figure comes with limited textual description and lacks a legend specifying what the different diagram elements mean. It is not even completely clear whether this is a Run-time or Functional view.

Another representative example is presented in Figure 3 from the openKM website[9]. This model depicts the user interfaces, security, main functionalities, and information storage of the system, mixing Functional (e.g., API/Distribution), Run-time (e.g., User Interface), and Deployment Views (e.g., Storage) in a single informal diagram. The colors and arrows in the model are not defined within a legend or supplementary text, which makes it difficult to interpret what they mean. It is also not clear whether this diagram is supposed to document the system at development time or execution time. Such a model is conflated, imprecise, and unclear, which limits its usefulness as software documentation. This also made it difficult to categorize diagrams for table III. The authors have decided that this model describes high level functionality, the types of devices/hardware that the users can use but doesn't give information about C-C patterns or the interaction flow.

According to the ISO/IEC/IEEE standard, views and models must address specific concerns for specific stakeholders. Most of the OSS projects in this study do not explicitly mention stakeholders with accompanying concerns. The authors of this work still find that the diagrams and general SA descriptions are still models as part of views, even if the relevant concerns and stakeholders are ambiguous or unknown. This is why table V denotes the inclusion of views and models even if stakeholders concerns are not explicitly identified.

The openEHR project is an odd one compared to the other five as it is not a typical monolithic software project. OpenEHR is a project encompassing standards, specification programs, clinical models, and software for the implementation of electronic health record systems. The project defines viewpoints and views, but since it operates on more of a meta-level, these are not the typical SA viewpoints and views. The nature of this project makes comparison to other projects harder.

VI. THREATS TO VALIDITY

This section addresses the threats to the validity of this study by following the case study research guidelines in [49]. Internal validity is not considered as no causal relationships were studied.

A. Construct Validity

This aspect of validity is concerned with the reasonableness of the used measures reflecting the intended construct. In this study, the threats to this validity are related to the extraction and classification of SA elements, as well as the determination of the maturity level. In order to mitigate the threats, the ISO/IEC/IEEE 42010:2011 standard was used to identify SA elements, this is an approach used in similar fashion in other works [3], [50]. In order to determine the maturity a proposed maturity model [25] was used, based on the same standard. Furthermore, the identification of SA elements and the application of the maturity model were done independently by the first two authors, discrepancies in the findings were

[6]https://doxy.audacityteam.org/class_audacity_command_dialog.html
[7]https://doxy.audacityteam.org/graph_legend.html
[8]https://developers.home-assistant.io/docs/architecture/core
[9]https://www.openkm.com/en/architecture.html

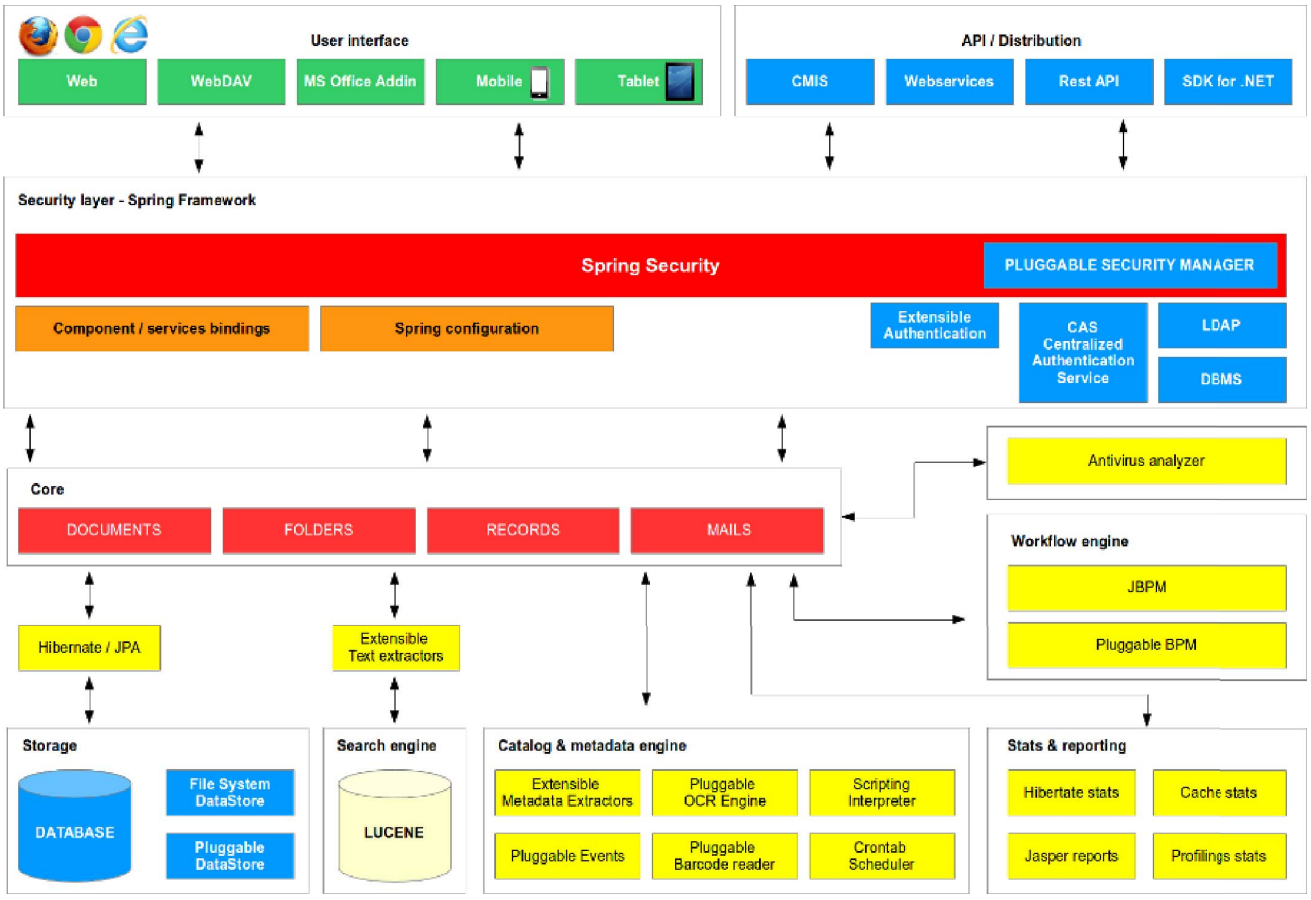

Fig. 3. System Architecture of openKM

resolved after a discussion. This was done to further reduce the possibility of personal bias or interpretation affecting the results.

B. External Validity

This aspect of validity is concerned with the degree to which the findings can be generalized. The observed results may not be applicable to other OSS projects. In order to mitigate this threat, six cases were selected operating in different domains, spanning an 18 year inception period. The cases were chosen to ensure a mixture of business, consumer, and research focus. The threat can't be fully ignored since it's likely that six diverse projects don't cover all OSS characteristics. Other remaining threats are related to the limited scope, gathering documents from the excluded sources may have lead to different results. Additionally, not being able to find documents is not necessarily indicative of their existence, e.g., core developers could have private documentation. Lastly, there is the threat of the possibility that documents that were publicly available were missed during the data collection. To mitigate this threat, the set of identified documentation was extensively reviewed by other members of the research lab beside the authors.

C. Reliability

This aspect of validity is concerned with the ability of other researchers to get the same results when this study is repeated. The main threats to reliability are related to the data collection and analysis process. In order to mitigate these threats, the research design uses existing models, standards, and theories publicly available in the open domain. Their sources and the steps taken in their application have been described so that other researchers can replicate this study. The main cause of differing results would be personal bias and interpretation as described under construct validity.

VII. Conclusion and Future Work

This study has investigated the SA documentation practices in OSS projects. The SA documents of six target projects were collected and analyzed to gain insight in what architectural elements are described, what document formats are used, what the formality of the description is, and what the maturity of the description is.

The main findings indicate that (1) most of the architecture description elements as prescribed by the ISO/IEC/IEEE 42010:2011 standard are not present in the SA description of OSS projects. The most commonly found elements are:

the system-of-interest, supplementary information, views, and models. (2) The found documentation addresses mainly functional aspects, descriptions regarding deployment were found the least. (3) Most documentation is available through web-based platforms like websites, wikis, or custom hyperlink content systems. One instance of a file-based approach for the recording of architecture decisions was found. (4) The formality of the SA description is overall informal with some semi-formal aspects. Natural language is widely used to describe SA. Informal models are also frequently used, yet their lack of notation increases ambiguity and inhibits their usefulness. Semi-formal diagrams are used by some projects, this includes UML and tool generated diagrams. Formal models employing frameworks or ADLs were not found. (5) The maturity of the SA descriptions in OSS is low. Out of five maturity levels, with level 5 being the highest maturity, only one project has a maturity of level 2, the other five projects do not get past level 1.

Future work could expand the scope and look at other data sources not analyzed in this study. The results show that the documentation is not comprehensive, informal, and ambiguous at times, yet some of the investigated projects have been running for many years successfully as can be derived from Sourceforge reviews, e.g., Audacity has currently 4.6 out of 5 stars[10], and download statistics, e.g., VLC has millions of downloads for multiple versions of their software[11]. The perception of contributors on the current documentation practices in their respective community could give additional insights into the documenting process, as well as the sufficiency of the amount and formality of available documentation. Additionally, a similar thing could be studied for new contributors to gauge if the state of the documentation is a relevant concern when deciding to join an OSS community. Regarding the maturity of SA description, the ArchCaMo maturity model was designed for SA in general. Further investigation could possibly devise a SA description maturity model that is more generically suited to OSS contexts, but also includes capabilities related to continuous development, product lines and variability, and quality concerns.

REFERENCES

[1] Open Source Initiative, "The Open Source Definition (Annotated)," 2007. [Online]. Available: https://opensource.org/osd-annotated

[2] O. S. Initiative, "Announcement of "OSI Certified" Open Source Mark," 1999. [Online]. Available: https://opensource.org/pressreleases/certified-open-source.php

[3] W. Ding, P. Liang, A. Tang, H. Van Vliet, and M. Shahin, "How do open source communities document software architecture: An exploratory survey," in *2014 19th International Conference on Engineering of Complex Computer Systems*, 2014, pp. 136–145.

[4] OpenUK, "State of Open: The UK in 2021 Phase Two: UK Adoption," OpenUK, London, Tech. Rep., Jul. 2021. [Online]. Available: https://openuk.uk/wp-content/uploads/2021/07/State-of-Open-Phase-Two.pdf

[5] Synopsys, "Open Source Security and Risk Analysis Report," Synopsys, Tech. Rep., Apr. 2021. [Online]. Available: https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-2021.pdf

[6] K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/libre open-source software development: What we know and what we do not know," *ACM Comput. Surv.*, vol. 44, no. 2, Mar. 2012. [Online]. Available: https://doi.org/10.1145/2089125.2089127

[7] J. Robbins, *Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools*. MIT Press, 2007, pp. 245–264.

[8] K.-J. Stol, M. A. Babar, P. Avgeriou, and B. Fitzgerald, "A comparative study of challenges in integrating open source software and inner source software," *Information and Software Technology*, vol. 53, no. 12, pp. 1319–1336, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095058491100142X

[9] V. K. Gurbani, A. Garvert, and J. D. Herbsleb, "A case study of a corporate open source development model," in *Proceedings of the 28th International Conference on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2006, p. 472–481. [Online]. Available: https://doi.org/10.1145/1134285.1134352

[10] J. Wesselius, "The bazaar inside the cathedral: Business models for internal markets," *IEEE Software*, vol. 25, no. 3, pp. 60–66, 2008.

[11] V. del Bianco, L. Lavazza, S. Morasca, and D. Taibi, "A survey on open source software trustworthiness," *IEEE Software*, vol. 28, no. 5, pp. 67–75, 2011.

[12] V. del Bianco, L. Lavazza, S. Morasca, D. Taibi, and D. Tosi, "An investigation of the users' perception of oss quality," in *Open Source Software: New Horizons*, P. Ågerfalk, C. Boldyreff, J. M. González-Barahona, G. R. Madey, and J. Noll, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–28.

[13] S. Morasca, D. Taibi, and D. Tosi, "T-doc: A tool for the automatic generation of testing documentation for oss products," in *Open Source Software: New Horizons*, P. Ågerfalk, C. Boldyreff, J. M. González-Barahona, G. R. Madey, and J. Noll, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 200–213.

[14] E. Y. Nakagawa, E. P. Machado de Sousa, K. de Brito Murata, G. de Faria Andery, L. B. Morelli, and J. C. Maldonado, "Software architecture relevance in open source software evolution: A case study," in *2008 32nd Annual IEEE International Computer Software and Applications Conference*, 2008, pp. 1234–1239.

[15] J. B. A. Jansen, "Software architecture as a set of architectural design decisions," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, 2005, pp. 109–120.

[16] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley Professional, 2012.

[17] R. H. Uwe Van Heesch, Paris Avgeriou, "Forces on architecture decisions – a viewpoint," in *2012 Working IEEE/IFIP Conference on Software Architecture*, 2012, pp. 101–110.

[18] W. Scacchi, "Free and open source development practices in the game community," *IEEE Software*, vol. 21, no. 1, pp. 59–66, 2004.

[19] P. Kruchten, "The 4+1 view model of architecture," *IEEE Software*, vol. 12, no. 6, pp. 42–50, 1995.

[20] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies: A survey," in *Proceedings of the 2002 ACM Symposium on Document Engineering*. New York, NY, USA: Association for Computing Machinery, 2002, p. 26–33. [Online]. Available: https://doi.org/10.1145/585058.585065

[21] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*, 2nd ed. Addison-Wesley, 2011.

[22] R. Plösch, A. Dautovic, and M. Saft, "The value of software documentation quality," in *2014 14th International Conference on Quality Software*, 2014, pp. 333–342.

[23] M. Reboucas, R. O. Santos, G. Pinto, and F. Castor, "How does contributors' involvement influence the build status of an open-source software project?" in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 475–478.

[24] R. Kazman, D. Goldenson, I. Monarch, W. Nichols, and G. Valetto, "Evaluating the effects of architectural documentation: A case study of a large scale open source project," in *IEEE Transactions on Software Engineering,*, vol. Volume 24, no. 3, 2016, pp. 220–260.

[25] A. A. C. Júnior, S. Misra, and M. S. Soares, "Archcamo - a maturity model for software architecture description based on iso/iec/ieee 42010:2011," in *Computational Science and Its Applications – ICCSA 2019*, S. Misra, O. Gervasi, B. Murgante, E. Stankova, V. Korkhov, C. Torre, A. M. A. Rocha, D. Taniar, B. O. Apduhan, and E. Tarantino, Eds. Cham: Springer International Publishing, 2019, pp. 31–42.

[10] https://sourceforge.net/projects/audacity/reviews/

[11] https://www.videolan.org/vlc/stats/downloads.html

[26] M. Levesque, "Fundamental issues with open source software development," *First Monday*, vol. 9, no. 3, October 2005. [Online]. Available: https://journals.uic.edu/ojs/index.php/fm/article/view/1484

[27] M. Michlmayr, F. Hunt, and D. Probert, "Quality practices and problems in free software projects," in *Proceedings of the first international conference on open source systems*, 2005, pp. 24–28.

[28] C. Holdgraf. (2017) Most developers think we should spend more time on documentation. [Online]. Available: https://bids.berkeley.edu/news/most-developers-think-we-should-spend-more-time-documentation

[29] R. S. Geiger, N. Varoquaux, C. Mazel-Cabasse, and C. Holdgraf, "The types, roles, and practices of documentation in data analytics open source software libraries," *Computer Supported Cooperative Work (CSCW)*, vol. 27, no. 3, pp. 767–802, 2018.

[30] A. Brown and G. Wilson, *The Architecture of Open Source Applications: Elegance, Evolution, and a Few Fearless Hacks*. Lulu.com, 2011, vol. 1.

[31] R. Hebig, T. H. Quang, M. R. V. Chaudron, G. Robles, and M. A. Fernandez, "The quest for open source projects that use UML: Mining GitHub," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. New York, NY, USA: Association for Computing Machinery, 2016, p. 173–183. [Online]. Available: https://doi.org/10.1145/2976767.2976778

[32] M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso, and G. Reggio, "Relevance, benefits, and problems of software modelling and model driven techniques—a survey in the italian industry," *Journal of Systems and Software*, vol. 86, no. 8, pp. 2110–2126, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121213000824

[33] S. A. Ajila and D. Wu, "Empirical study of the effects of open source adoption on software development economics," *Journal of Systems and Software*, vol. 80, no. 9, pp. 1517–1529, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121207000076

[34] C. Seaman and Y. Guo, "Chapter 2 - measuring and monitoring technical debt," in *Advances in Computers*, M. V. Zelkowitz, Ed. Elsevier, 2011, vol. 82, pp. 25–46. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123855121000025

[35] N. Rios, L. Mendes, C. Cerdeiral, A. P. F. Magalhães, B. Perez, D. Correal, H. Astudillo, C. Seaman, C. Izurieta, G. Santos, and R. Oliveira Spínola, "Hearing the voice of software practitioners on causes, effects, and practices to deal with documentation debt," in *Requirements Engineering: Foundation for Software Quality*, N. Madhavji, L. Pasquale, A. Ferrari, and S. Gnesi, Eds. Cham: Springer International Publishing, 2020, pp. 55–70.

[36] A. Baabad, H. B. Zulzalil, S. Hassan, and S. B. Baharom, "Software architecture degradation in open source software: A systematic literature review," *IEEE Access*, vol. 8, pp. 173 681–173 709, 2020.

[37] M. Riaz, M. Sulayman, and H. Naqvi, "Architectural decay during continuous software evolution and impact of 'design for change' on software architecture," in *Advances in Software Engineering*, D. Ślezak, T.-h. Kim, A. Kiumi, T. Jiang, J. Verner, and S. Abrahão, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 119–126.

[38] D. M. Le, D. Link, A. Shahbazian, and N. Medvidovic, "An empirical study of architectural decay in open-source software," in *2018 IEEE International Conference on Software Architecture (ICSA)*, 2018, pp. 176–17 609.

[39] D. Rost, M. Naab, C. Lima, and C. von Flach Garcia Chavez, "Software architecture documentation for developers: A survey," in *Software Architecture*, K. Drira, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 72–88.

[40] K. de Graaf, P. Liang, A. Tang, and H. van Vliet, "How organisation of architecture documentation affects architectural knowledge retrieval," *Science of Computer Programming*, vol. 121, pp. 75–99, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016764231003172

[41] "Systems and software engineering — architecture description," International Organization for Standardization, Geneva, CH, Standard ISO/IEC/IEEE 42010:2011, 2011. [Online]. Available: https://www.iso.org/standard/50508.html

[42] R. Caralli, M. Knight, and A. Montgomery, "Maturity models 101: A primer for applying maturity models to smart grid security, resilience, and interoperability," Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, Tech. Rep., 2012. [Online]. Available: https://resources.sei.cmu.edu/asset_files/WhitePaper/2012_019_001_58920.pdf

[43] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical software engineering*, vol. 10, no. 3, pp. 311–341, 2005.

[44] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "An exploratory study on confusion in code reviews," *Empirical Software Engineering*, vol. 26, no. 1, pp. 1–48, 2021.

[45] D. Garlan, R. T. Monroe, and W. David, "Acme: An architecture description interchan language," in *Proceedings of CASCON'97*, Toronto, Ontario, November 1997, pp. 169–183.

[46] R. Allen, "A formal approach to software architecture," Ph.D. dissertation, Carnegie Mellon, School of Computer Science, January 1997.

[47] N. Medvidovic, R. N. Taylor, and E. J. Whitehead Jr, "Formal modeling of software architectures at multiple levels of abstraction," *Proceedings of the California Software Symposium 1996*, pp. 28–40, 1996.

[48] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation: understanding the decisions of open source contributors," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 127–136.

[49] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering volume*, vol. 14, no. 131, 2009. [Online]. Available: https://link-springer-com.proxy.library.uu.nl/article/10.1007/s10664-008-9102-8

[50] T. Bi, W. Ding, P. Liang, and A. Tang, "Architecture information communication in two OSS projects: The why, who, when, and what," *Journal of Systems and Software*, vol. 181, p. 111035, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221001321