Architectural Views: The State of Practice in Open-Source Software Projects

Sofia Migliorini[1], Roberto Verdecchia[1], Ivano Malavolta[2], Patricia Lago[2], and Enrico Vicario[1]

[1] University of Florence, Italy
sofia.migliorini@edu.unifi.it,{roberto.verdecchia,enrico.vicario}@unifi.it
[2] Vrije Universiteit Amsterdam, The Netherlands
{i.malavolta,p.lago}@vu.nl

Abstract. Context: Architectural views serve as fundamental artefacts for designing and communicating software architectures. In the context of collaborative software development, producing sound architectural documentation, where architectural views play a central role, is a crucial aspect for effective teamwork. Despite their importance, the use of architectural views in open-source projects to date remains only marginally explored.
Goal: We aim at conducting a comprehensive analysis on an extensive corpus of open-source architectural views. The goal is to understand (i) what the "history" of architectural views is, (ii) how architectural views are represented, and (iii) what architectural views are used for in the context of open-source projects.
Methods: We leverage a software repository mining process to systematically construct a dataset of 15k architectural views. Then, we perform (i) a quantitative analysis on the metadata of all 15k views and (ii) a qualitative analysis on a statistically-relevant sample of 373 views.
Results: Most projects rely on a single architectural view, which is often used to document a medium or high level description of the architecture. Views are usually created at either the beginning or at the end of a project, are rarely updated, and tend to be maintained by a single contributor. Views usually adopt an informal colored notation without a supporting legend and frequently report technologies used. Deployment and control flow are the most recurrent viewpoints, and commonly cover concerns related to software maintainability and functional suitability.
Conclusion: The state of the practice about architectural views in open-source software systems seems to favor informal descriptions. Despite this, the effort needed to create views might hinder keeping views up to date, and a common syntactic ground between viewpoints seems hard to find. To address current needs, we speculate that a solution could lie in defining and popularizing versionable, templateable views that can be integrated in collaborative programming environments.

Keywords: Architectural Views · Architectural Documentation · Repository Mining · Open Source Software

1 Introduction

In the vast landscape of software development, architecture plays a key role as a bridge between requirements and implementation [4]. A robust architecture has the potential

to guarantee that a system will meet essential quality requirements in such areas as performance, reliability, portability, scalability, and interoperability [8]. One of the primary methodologies to design and communicate software architectures is through the use of *architecture viewpoints* and *architectural views* [6]. Different stakeholders generally have drastically differing mental models based on their experiences with a software-intensive system, and may only primarily focus on certain aspects of it [27]. By quoting Rozanski and Woods: "*A view is a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders* [23]. A software architecture is a complex entity that defies a basic one-dimensional description. With the help of views, which enable the separation of concerns, we can break down the multidimensional structure into a number of, hopefully, engaging and comprehensible system representations [4].

In the domain of collaborative software development, GitHub is one of the leading platforms for open-source software projects [11]. With 420 million total projects at the end of 2023[3], GitHub actively promotes collaboration, enables transparent communication, and empowers developers to contribute to open-source projects globally. Although documentation is essential to the open-source ecosystem, as it helps developers learn about projects and, consequently, choose which ones to contribute to [31,28], open-source documentation is frequently regarded as inadequate, sparsely written, outdated, or nonexistent [9]. In the year 2000, IEEE introduced a standard for architecture documentation (the latest version updated in 2022 [2]) that advocates creating personalized views that best address the stakeholders and their concerns associated with the system to be represented. Although there are studies to identify different types of documentation provided in software repositories [21,12], none of them focuses on software architecture views. This work aims to fill this void by focusing on an extensive corpus of architectural views, intending to investigate their adoption within the context of open-source projects.

The main contributions of this study can be summarized as follows:

- A publicly-available dataset of 15k software architecture views extracted from 12.2k GitHub projects, supported by repository and view commit history metadata.
- A quantaitive analysis of the collected data, casting light on the practice of software architecture views in open-source practice.
- An in-depth manual categorization of a statistically representative sample of 373 views, conducted by studying both their syntactic and semantic properties.
- A comprehensive replication package of the study (see Section 8).

2 Related work

To the best of our knowledge, this study represents the first analysis of an extensive corpus of software architecture views extracted from open-source projects.

The most similar large-scale study, conducted by Hebig *et al.* [12], focuses on the analysis of UML usage in open-source projects, involving a systematic mining of GitHub repositories. In addition, Buchgeher *et al.* [5] proposed a study on the adoption of architecture decision records (ADRs) in projects hosted on GitHub. Both studies aim at characterizing current practices of file creation and maintenance over time of either UML models or ADRs, without specifically focussing on software architecture views. Our study extends the research by also analyzing the visualization techniques and contents of software architecture views, without being confined to a specific language or format.

[3] `https://github.blog/2023-11-08-the-state-of-open-source-and-ai`

Expanding the perspective, Ding *et al.* [7] proposed a comprehensive investigation of 2k projects from four major open-source software sources, identifying 108 projects with documented software architecture. Their goal is to analyze what type of information is documented and how it is described. Similarly, Muszynski *et al.* [18] examine software architecture documentation practices in six large popular open-source software systems; the focus of this study is the classification of architectural views. Malavolta *et al.* [17] analyzed 335 open-source robotics projects and mined how roboticists document the software architecture of their system, which types of views are used and how they are represented (*e.g.,* textually or visually). Unlike these studies, which examine a limited number of open-source projects, our investigation targets architectural views extracted from 12.2k repositories. Furthermore, instead of presenting a general overview, we focus on a particular type of software architecture documentation, namely software architecture views.

A classification of software architecture visualization techniques reported in the literature is presented by Shahin *et al.* [25]. A similar review of recent and key literature on software architecture visualization is performed by Ghanam and Carpendale [10]. Additionally, Alshuqayran *et al.* [3] presented a systematic mapping study on microservices architectures and their implementation, offering insights on architectural views/diagrams used. Rather than focusing on visualization techniques, our study aims at classifying software architecture views by analyzing both the languages syntax and views contents. In addition, we based our research on the practical use of software architecture views in open-source repositories, instead of relying on exsisting literature.

Several surveys have been conducted to understand the perspectives of practitioners on software architecture documentation and related tools. Rost *et al.* [22] investigated problems and wishes for the future with industrial participants. Ozkaya [19] studied the level of knowledge and experience in software architectures among practitioners from both industry and academia. Malavolta *et al.* [16] examined the use of architectural languages in industry by interviewing practitioners from 40 different IT companies. Differently from such surveys, our research does not focus exclusively on industrial practices, but rather extracts data from GitHub repositories, characterizing the state of practice of software architecture views in the context of open-source software projects. Moreover, the main results of these surveys focus on architectural notations; our research also covers the contents of architecture documentation practices.

A more specific survey by Ozkaya and Erata [20] aims at understanding the usage of UML diagrams for modeling from different viewpoints. Additionally, a case study on the selection of viewpoints in projects from three different telecom-area software organizations was held by Smolander [26] to clarify how the conceptions of architects about architectural viewpoints differ, based on the prevalent situation and characteristics in an organization and in the project at hand. Unlike these studies that focus on a particular architectural language or a limited number of case studies, our analysis encompasses the usage of viewpoints across numerous software architecture views, without relying on any specific architectural language.

3 Research Methodology

3.1 Goal and Research Questions

The goal of this research is to characterize the state of practice of software architecture views in open-source projects. This study aims to answer, in the context of open-source software, the following research questions:

RQ1: *What is the history of architectural views?* This research question aims at understanding the timing of views introduction in repositories, how they evolve over time, and who contributes to this evolution.

RQ2: *How are architectural views represented?* With this question we aim at identifying the main characteristics of the notations used to describe architectural views.

RQ3: *What architectural information has been represented in the views?* This research question constitutes the core of the study. By answering it, we aim at characterizing our view dataset in terms of topics focused on, quality requirements considered, architectural styles, and technologies employed.

3.2 Research Process

Figure 1 shows an overview of the research process of this study. The details of each step are provided in the remainder of this section.

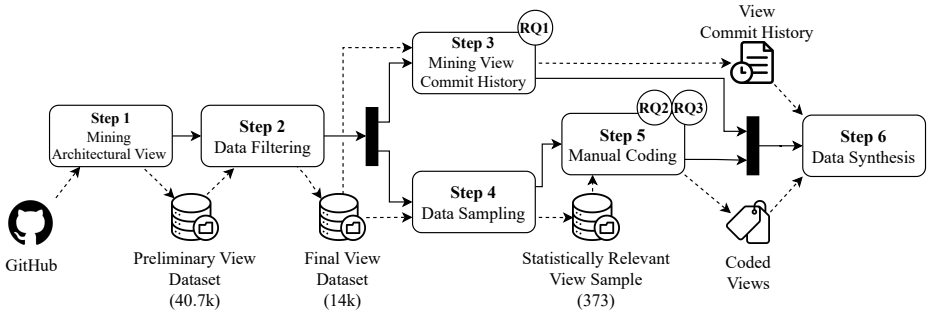


Fig. 1: Research Process Overview

Step 1: Mining Architectural Views. The objective of this mining step is to identify on GitHub images reporting an architectural view. In order to do so, we first automatically identify all files containing the substring "architect" in the filename that are linked in any of the `README.md` and `CONTRIBUTING.md` files hosted in a GitHub repositories (including the ones located in subfolders). This process is executed by sending requests to the Search Code endpoint of the GitHub REST API[4]. Below a list of the restrictions applied for this search:

- R1: Consider only the *default branch* of the repository,
- R2: Search for files smaller than 384 KB,
- R3: Search for repositories with fewer than 0.5M files,
- R4: Search for repositories that have had activity or have been returned in search results in the last year,
- R5: Do not include repository forks.

R1-R4 are forced by the Search code endpoint[5], whereas R5 is a choice that we make for our study in order to avoid duplication. Regarding R2, we observe that this limitation does not compromise our data collection as the file retrieval stops long before reaching the imposed limit. The mining phase was carried out between September and October 2023. For each extracted URL, we also gather relevant metadata about the repository that

[4] `https://docs.github.com/en/rest`. Accessed 9th April 2024.

[5] `https://docs.github.com/en/rest/search/search?apiVersion=2022-11-28#search-code`. Accessed 9th April 2024

contains the file. At the end of this extraction phase, a duplicate removal is performed based on the image URL, and a dataset of 40.7k architectural views is produced.

Step 2: Data Filtering. After the first mining phase, a filtering process follows to ensure the quality of the mined views. The exclusion criteria applied are the following:
- E1: Views or repositories referring in their name to demos, examples, or exams,
- E2: Views contained in a repository with less than 10 commits,
- E3: Views contained in a repository with less than 2 stars,
- E4: URLs linking to GitHub badges, such as "img.shields.io",
- E5: Non-downloadable images.

E1 serves to target real projects: We define a set of regular expressions and a view is discarded if either the repository name, the repository description, or the image URL matches any regular expression [17]. E2 and E3 are used to avoid inactive or non-maintained projects [17,15]. From a quick sample analysis of the dataset, we also identify a set of URLs that point to some GitHub badges, which are removed from the dataset (E4). Finally, we exclude views that produce an error in the downloading phase, pointing to views that are no longer available (E5). At the end of this filtering phase, we establish a dataset comprising 15k views. An example of view developed by the Google Cloud Platform[6] taken from the dataset is documented in Figure 2.

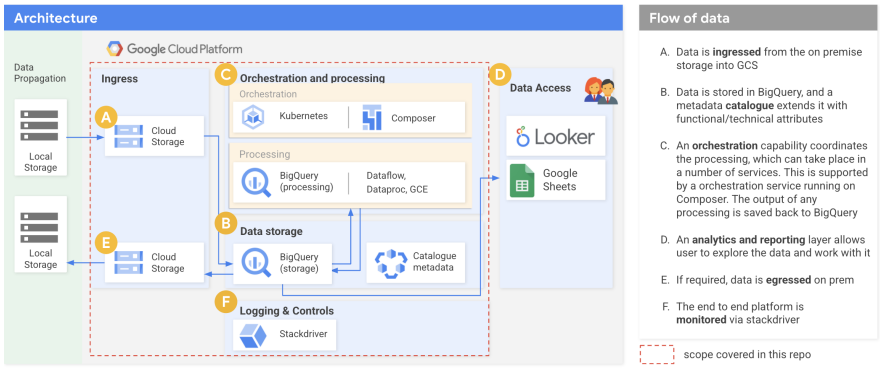


Fig. 2: Example of Mined Architectural View[6]

Step 3: Mining View Commit History. For each view file we extract (i) the number of commits, (ii) the number of distinct contributors, (iii) the number of days between the first commit of the repository and the first commit adding the file representing the view, (iv) the total number of commits prior to the introduction of the view file, and (v) the number of commits between the first and last view update. The commit history is extracted for 14.2k views, as 6% of identified views resulted to be inaccessible (*e.g.,* images no longer in the repository, spelling errors, path errors). The data collected in this step is used to answer RQ1.

Step 4: Data Sampling. In order to answer RQ2 and RQ3 we need to extract a statistically significant sample from our dataset. For this purpose, an additional data fil-

[6] `https://github.com/GoogleCloudPlatform/reg-reporting-blueprint`

tering step is performed to consider exclusively one architectural view for each repository, leading to selecting 12.2k views. With the dataset of 12.2k views, to ensure 5% margin of error and 95% confidence level, we randomly sample 373 views. With a preliminary analysis of the first extracted sample, we can conclude that our dataset presents about 23% of false positives. In particular, the main categories of rejected views represent: the internal architecture of machine learning algorithms (*e.g.,* the layers of a deep neural network), with 59 views (15.8%), hardware architectures (9 views, 2.5%), network architectures (5 views, 1.3%), and logos (5 views, 1.3%). Views containing text in languages other than English are retained and automatically translated for subsequent analysis.

Step 5: Manual Coding. Three data exploration sessions follow, where researchers discuss the characteristics of the views that are relevant to answering RQ2 and RQ3, leading to the consolidation of the data extraction framework used for this study. The sample dataset is then thoroughly analyzed *via* iterative coding sessions. For scrutiny and replication purposes, detailed explanations of the codes used as coding guide are available in the replication package (see Section 8).

Given that RQ2 addresses the syntax of the visualization techniques employed, to answer it, we extract the following fields: (i) architectural notation employed, *i.e.,*, informal, semiformal or formal [6] (ii) shapes used, *e.g.,* rectangles or circles, (iii) use of color, (iv) presence of a legend, (v) presence of nested components, (vi) presence of explicit ports/interfaces between components, (vi) presence of explicit connectors, *i.e.,* lines connecting components, and (vii) connectors direction, *i.e.,* unidirectional, bidirectional, non-explicit or bus.

In order to answer RQ3, we take a step further and proceed to study the content of the views by considering the following fields: (i) architecture scope, *i.e.,* if the view represents a part of the system, the entire system, or the entire system plus how it interfaces with other systems, (ii) architectural style(s) used in the view, (iii) concern(s) addressed, *e.g.,*, deployment or connectivity, (iv) behavior, *i.e.,* if the view represents static, dynamic properties of the architecture, or both, (v) quality attribute(s) considered (first level of the *ISO/IEC 25010:2023* classification [13]), (vi) granularity of the system representation, *i.e.,* high, medium, or low, (vii) components nature, *e.g.,* servers or databases, (ix) connectors nature, (x) technologies reported (classification at the level of the software provider, *e.g., AWS* and not *AWS Lambda*), and (xi) design overlays, *e.g.,* textual descriptions or code snippets [1].

Regarding the coding of *Architecture Scope* and *Granularity*, we use magnitude coding in order to capture the intensity of the feature considered. We use provisional coding [24] to classify the *Quality Attributes*, *Behavior*, *Connectors Direction*, and all the *yes/no* fields. For all other fields, we use open coding [14] to identify recurrent concepts. Two researchers independently analyzed 186/187 views, with weekly discussions used to revise, homogenize, and align the coding process. When necessary, doubts and disagreement points were resolved by involving a third researcher. The data gathered through this step is used to answer RQ2 and RQ3.

Step 6: Data Synthesis. To answer RQ1, which is purely quantitative, the extracted data are analyzed and interpreted by simple statistical means, such as data plotting and basic summary statistics. To answer RQ2 and RQ3, the codes obtained from the labeling process are first examined to see if any can be removed or merged with others due to a lack of representativeness. Then, the resulting ones are presented with different plots according to the information that we want to describe. For the discussion section,

a phase of cross-matching results is added to extract potential interesting relationships among the various aspects analyzed.

4 Results

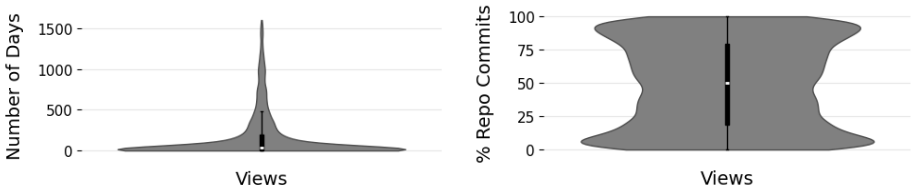
In this section, we present the results collected to answer our RQs (see Section 3.1).

4.1 Results RQ1: History of Architectural Views

Architectural Views Creation. Figure 3(a) illustrates the moment of the view introduction in the repository in terms of days passed since the start of the project, which is represented by the initial commit in the repository. Outliers are identified and excluded from the plot to enhance visualization. Our collected data reveals a tendency to introduce architectural views at the project start: the project's age when the view is introduced is on average 222.51 days, with a median of 39 days, while projects have an average duration of 1570.19 days, with a median of 1456 days.

The maximum value (9316 days) is recorded for the repository *PolarDB*[7], a cloud-native database developed by *Alibaba Cloud* active since 1996, which is characterized by 56 contributors, 2.6k stars and 427 forks. At the beginning, the repository contained the source code distribution of the *PostgreSQL* database management system. In 2021, it changes its subject to *PolarDB* and a complete new `README.md` file is introduced by a new contributor, including an architectural view.

However, the level of activities can highly vary throughout the lifespan of open-source projects. In order to place the view introduction relatively to other development activities of the project, Figure 3(b) illustrates the distribution of views based on the percentage of repository commits done when the view is introduced. This figure presents a much more balanced perspective, showing that architectural views are introduced in repositories along all active phases of the projects, with peaks in the initial and final ones. The median and mean of the percentage of overall commits done when the view is introduced are 50% and 49.34%, respectively.



(a) *Project's age in days at the moment of the view introduction in the repository (without outliers)*

(b) *Percentage of overall repository commits done when the view is introduced*

Fig. 3: Views Creation

Architectural Views Maintenance. Our results show that the vast majority of views with a retrieved commit history (10.6k out of 14.2k) remains unchanged. Yet, we found that around 25% of them (3.5k out of 14.2k) are updated one or more times.

[7] `https://github.com/ApsaraDB/PolarDB-for-PostgreSQL` Accessed 14th April 2024.

Figure 4 summarizes the distribution of views by number of commits. Outliers are excluded from the plot for a better visualization.

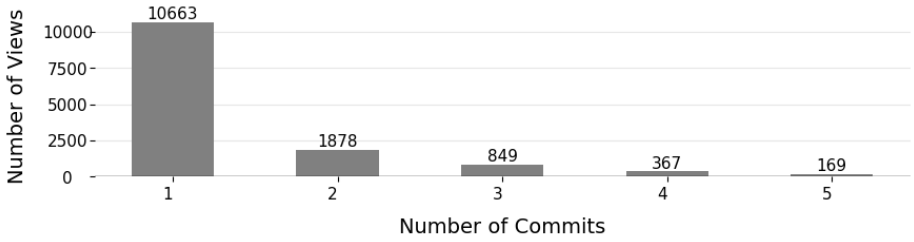
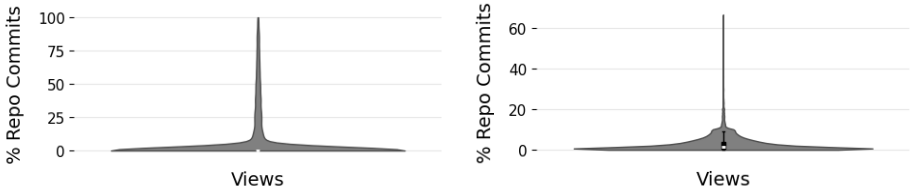


Fig. 4: Distribution of views by number of commits (without outliers)

When a view is updated, the number of commits is on average 3 (2 updates), with a median of 2 (1 update). The maximum number of commits on a view is 27 for a view that is updated over time by the same user[8], even though the repository has a total of seven contributors.

We have also conducted an analysis of the time span between the first and last commit targeting a view within a repository. This time span, shown in Figure 5(a), is expressed as a percentage of overall commits of the repository. The maximum time span observed is 100% of repository commits, while the mean time span is 6.98%. The majority of projects seem to concentrate the view introduction and updating activities within a relatively short period, with a median of 33 days, without interspersing them with other substantial development tasks. The high frequency of 0% cases is due to the predominance of views with just 1 commit. Updating activities extend beyond 10% of the project's commits only in a minority of repositories.



(a) *Time passed between first and last view commit expressed as percentage of overall repository commits*

(b) *Percentage of overall repository commits dedicated to the view updates*

Fig. 5: Views Maintenance

Figure 5(b) shows a focus on the percentage of repository commits dedicated to the view creation and updating, with a mean of 2.91% and a median of 1.54%. The maximum of 66.67% is reached for a repository that has exclusively the purpose of describing the documentation of the project[9].

[8] `https://github.com/digitaltwinconsortium/ManufacturingOntologies/blob/main/Docs/architecture.png`. Accessed 15th April 2024.

[9] `https://github.com/gridsuite/documentation`. Accessed 15th April 2024.

Contributors. More than 92% of views (13.2k out of 14.2k) have just one contributor, while the remaining 8% are updated by more than one user, with a maximum of 5. In particular, 72% of views that have at least one update are modified by the same contributor.

An additional analysis is conducted on the percentage of contributors editing the view. The median and mean are 50% and 57% of total contributors, respectively, that is due to the presence of many repositories with just 1 contributor. The minimum is 0.22% of repository contributors.

> **RQ1: Architectural Views History.** Architectural views tend to be introduced either at the beginning or at the very end of open-source projects. 75% of views are never updated. Activities on views are concentrated in short periods, without interspersing them with other significant development tasks. When updated, 71% of views are edited by the same user.

4.2 Results RQ2: Architecture Representation

Architectural Notation. Out of the statistically relevant sample considered to answer RQ2 (373 views), the vast majority of views use *informal* notations (96%), notably *boxes and lines*. Only a much smaller fraction (4%) employs a *semiformal* notation, in particular *UML*, mainly focussing on *low-level* aspects of the architecture. There is no evidence for the use of formal notations. Out of the views utilizing informal notations, seven resulted to be manual sketches.

Shapes. Figure 6 presents a distribution of views in terms of what kind of shapes are used in the views. The most popular shapes employed are *rectangles* (79%), and *icons* (46%), which mostly depict the icon of a technology. These *primary shapes* mainly appear as the only shape in the view, whereas all other shapes are always combined with each other or used as auxiliary shapes for specific elements, *e.g.,* cylinders for databases.

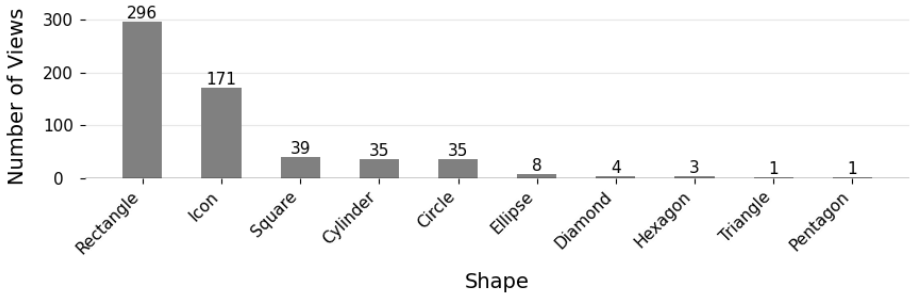


Fig. 6: Number of views *per* shape

Architectural views are mainly mapped to more than one shape. We observe the presence of a small group of 3-dimensional images (2%) generated by the Cloudcraft platform[10], which appears to be used also in a considerable number of bidimensional views.

Connectors. Most views use explicit connectors between architecture elements (92%). Different types of connectors may coexist in the same view, but we observe that when

[10] `https://www.cloudcraft.co`. Accessed 15th April 2024.

connectors are used, their direction is mostly *unidirectional* (74%). The presence of *bidirectional* connectors is detected in almost a third of the views (29%). We remark that sometimes bidirectional connections are shown as two unidirectional ones. Only three views present a bus (1%), while few more do not show explicit connector directions (5%).

Further Properties. Regarding further view properties, we noted that 81% of views are colored, even if the specific meaning of colors appears to be seldom defined, both for components and connectors. Only 8% of views include a legend to provide further explanations about the employed notations.

In terms of structural complexity, the use of nested components is slightly more frequent than unidimensional representations (56%), indicating a preference for hierarchical organizations within architectural views.

Regarding the presence of explicit ports or interfaces between architectural components, only a fraction of views show them (8%).

RQ2: Architectural Views Syntax. Most views use an informal notation (96%), while only a minor portion (4%) a semiformal one (UML). Rectangles and icons are the most widely used shapes. Unidirectional connections are the most represented (74%). Views often use colors (81%), but only 7% of them include a legend documenting the color use. Explicit ports or interfaces between components are rarely documented (8%).

4.3 Results RQ3: Architectural Views Contents

Architecture Scope. Regarding the representation level of the architectural views, in most cases, the *entire* architecture of the system is represented (53%), instead of focusing only on a *part* of it (24%). A considerable fraction of the considered views (23%) also documents *interactions* between the system considered and other external systems.

Architectural Styles. The most recurrent architectural styles documented in the views are *client-server* (20%), *layered* (20%), *service-oriented* (15%), and *event-driven* (12%). Out of the views reporting service-oriented architectures, 35% of them focus specifically on *microservices*. Event-driven architectures instead often result to manage events with serverless functions, most notably AWS Lambda[11].

Concerns. Regarding the topics addressed by architectural views, we note various recurring subjects. Views can be mapped to more than one concern, with 20% of the total covering simultaneously more than one concern. A considerable portion of the views focus on *general* architectural documentation (30%), *i.e.,* rather than focusing on a specific topic, they adopt a broad and neutral technological viewpoint.

The second most recurrent concern covered by views is *deployment* (30%), with a recurrent focus on cloud and virtualization aspects. Following closely are *control flow* views (29%), used to document functional aspects of architectures. *connectivity* among architecture components and sub-systems is another recurrent concern (16%), followed by *data flow* (8%), and a smaller number of *security* views, considering mostly message cryptography and authentication mechanisms. *Performance* concerns are depicted only 3% of the views, and often illustrate architectural aspects related to load balancers and distributed systems. Finally, architectural viewpoints regarding *scheduling* are adopted in 1% of the views.

[11] `https://aws.amazon.com/lambda`. Accessed 10th April 2024

Granularity. In terms of granularity of the system representation, the majority of views exhibit either a *high* granularity one (53%), reporting a very coarse architectural representation, *e.g.,* just the architectural layers, or a *medium* granularity (37%), *e.g.,* considering the main sub-systems and key technologies used. A much smaller portion of views use a *low* level of granularity granularity (9%), and are mostly used to reason about source code implementation details, such as functions or attributes in addition to broader system components.

Behavior. Regarding the architecture behavior depicted, approximately half of the views consider *static* properties of architectures (49%). A smaller number instead considers *dynamic* properties (42%), such as the system *control flow*, are represented in a smaller portion of views. Finally, a subset of views considers both static and dynamic aspects (9%).

Quality Attributes. *Maintainability* is the most recurrent quality attribute considered in the views (68%), reflecting the high recurrence of the *general* concern of the views previously documented. *Functional suitability* is the second most recurrent concern (32%), and is mostly mapped to viewpoints detailing step-by-step use case executions through architectures. Least addressed QAs are instead *performance efficiency* (35%), *security* (9%), *flexibility* (8%), interaction capability (7%), compatibility (2%) and reliability (2%). As for the concerns category previously presented, a view may address more than one QA. Out of all ISO/IEC 25010 characteristics, *safety* is the only quality attribute that is not considered in the views.

Components Nature. Regarding the nature of components represented in the views, Figure 7 documents the frequency of the different identified components across views. The most frequent components are *technologies* (present in 36% of the views). Other recurrent component types are, in order of frequency, *databases* (32%), *sub-systems* (24%), *clients* (24%). The *others* category reported in Figure 7 includes components utilized only in one or two views, *e.g.,* virtual machines, firewalls, and coroutines.

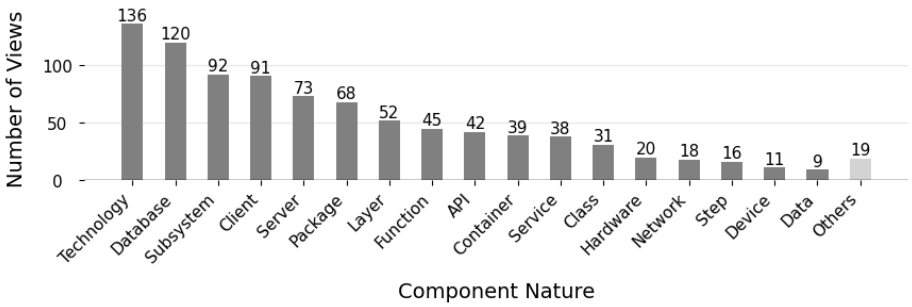


Fig. 7: Frequency of the different components across views

Connectors Nature. Regarding the interpretation of explicit connectors, it was possible to classify their nature in 91% of the dataset. In particular, 8% of views do not show explicit connectors, while for 1% of them the meaning of connectors does not emerge from the views' analysis. *Control flow* are the most used connectors type (28%), and are frequently used in combination with numerical indicators highlighting sequential operations transversing architectures at runtime. General *communication* among components, frequently used to highlight communication channels or protocols,

are also recurrent (27%). Views with *data flow* connectors are a bit less frequent (25%), and commonly co-appear in views along database components. Least recurrent connector types are dependencies (12%), function calls (5%), inheritance dependencies (3%), API calls (2%) and composition relations (1.3%). As for other studied properties, a single view may incorporate multiple types of connectors, reflecting the complex interactions and relationships within the software system.

Technologies. Figure 8 documents the ten most frequently mentioned technologies in the architectural views[12].

The overall high recurrence of cloud-based and virtualization technologies, *e.g.,* Amazon Web Services (16%), Kubernetes (7%), and Docker (7%), may be attributed to the current growing adoption of cloud-native serverless applications.

Reporting at least a technology in the views results to be a rather widespread practice (35%). The median number of reported technologies per view is one and the mean is two, while the maximum number of technologies reported in a single view is ten.

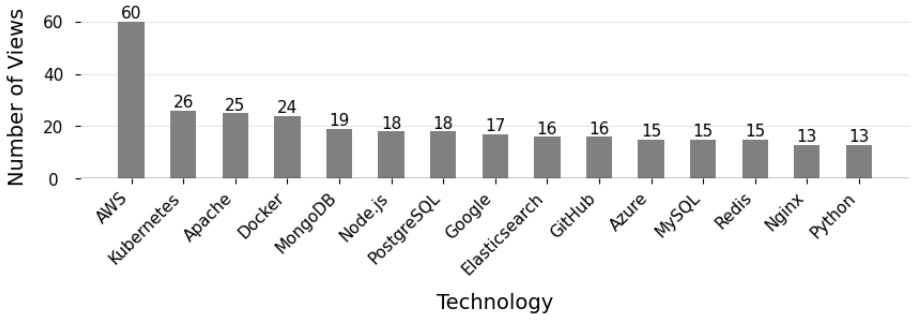


Fig. 8: Number of views per technology (top 10)

Design Overlays. Additional information to support the views is utilized in approximately a fifth of all views (18%). The majority of the overlays report complementary architectural information in form of *textual descriptions* (45% of views using overlays), which is primarily used to specify system or components functionalities. When *screenshots* are employed (17%), they mainly depict the user interface of an application or website. *miniviews* are instead less frequently used to zoom-in on particular aspects of the architecture (12%), *e.g.,* to document a data transformation process. Other information such as *URLs* (10%), *code snippets* (9%), and *configuration parameters* (7%) are also, but less frequently, used to highlight connection or low-level implementation details.

[12] The complete list of technologies and their recurrence is reported for completeness in the replication package of this study

RQ3: Architectural Views Content. 75% of views consider architectures in their entirety. Client-server is the most used architectural style (25%), followed by layered (20%), service-oriented (15%) and event-driven (12%) architectures. General architecture documentation, followed by deployment and control flow are the most recurrent topics covered. Views mostly consider a high or medium granularity level (91%). Static and dynamic aspect are equally represented. The most considered quality attribute is maintainability (68%), followed by functional suitability (33%). Around 65% of views explicitly report the technologies used, with AWS being the most recurrent one (25%).

5 Discussion

From the data collected a clear picture regarding the use of views in open-source practice emerges. *Most software projects rely on a single, seldom updated view, which mostly utilizes an informal notation to outline the high level architectural structure.* A common syntactic ground among viewpoints is very hard to be found.

Despite the importance of views for software documentation [23], their intuitive support for onboarding processes, and their potential to stimulate new contributions in collaborative environments such as GitHub, architectural view documentation in the open-source community does not seem to be a well established practice. As step forward, we ask ourselves: *What can we do to ease the adoption of architectural views in open-source practice, and how can we systematize the documentation of views?*

A potential answer to this question lies in *making views versionable*. In fact, to date, the process required to create and update architectural views on GitHub seems quite cumbersome. More specifically, editing an architectural view commonly relies on utilizing external tools which are not integratable with GitHub, and hence require first to update the view *via ad-hoc* standalone tools, and subsequently push an unmodifiable version of the view in shared repository. In addition to making views versionable, another aspect crucial for the adoption of view documentation in open-source could be to *allow views to be automatically rendered in the web interfaces of repositories*, in a similar fashion with which `README.md` files are currently visualized on GitHub websites. This would further support to move away from hard-coded immutable view image files in open-source repositories, while still providing a swift and intuitive graphical representation of views. Singular efforts striving to provide versionable web-friendly views, such as Mermaid[13], which allows to create various versionable diagrams on GitHub, *e.g.,* C4 and UML, should be therefore incentivized and given more visibility. Providing support and exposure to such projects would allow to move view documentation in open-source from a small community endeavors to mainstream, consolidated, and well integrated open-source practices. As related consideration, similar to current trends pushing for the standardization of architectural decision records in open-source[14], the community could spend efforts to *establish reusable templates to represent architectural views.*

We speculate that providing versionable, templateable views that can be integrated in collaborative programming environments would allow to further establish architectural view documentation practices in open-source. Through virtuous cycles, this could lead to further cascading effects, *e.g.,* the inclusion of GitHub Actions[15] dedicated to software architecture, and the rise of dedicated `ARCHITECTURE.md` documentation.

[13] `https://github.com/mermaid-js/mermaid`. Accessed 24th April 2024.
[14] `https://adr.github.io`. Accessed 24th April 2024.
[15] `https://github.com/features/actions`. Accessed 24th April 2024

6 Threats to Validity

We document the study threats by following to the categorization proposed by Wholin *et al.* [30] and considering common pitfalls in assessing threats to validity [29].

Construct Validity. To ensure that the dataset of views addresses our research questions, we defined *a priori* the concept of *software architectural view* by relying on established definitions [2,6]. The entirety of the study is conducted by following such definition. *Via* a set of *a priori* defined inclusion criteria used during the mining phase, and a subsequent manual process based on a coding guide, views deviating from our construct definition were discarded to ensure adherence of the collected results to our research goal.

Internal Validity. A potential internal validity threat lies in the formulation of the search query used in the mining process (see Section 3, Step 1). To mitigate this threat, the query was refined over more iterations through a set of exploratory query trials. Another potential internal threat regards subjective biases that could have influenced the manual coding process (see Section 3, Step 6). To mitigate this threat, two annotators were involved in the process, weekly meetings were held to align the labeling process by discussing doubts and examples, and a third researcher was involved whenever necessary. The automated translation accuracy of non-English views could have also potentially influenced the internal validity of the study. However, given their low recurrence (2%), we do not deem it could have majorly influenced our results. Exclusion criteria E1 (see Section 3.2) could had led to the inclusion of demo projects in our dataset. However, as no toy project was found in the manually scrutinized sample (see Section 3.2), we do not deem this threat majorly influenced our results.

External Validity. Albeit our best efforts, the automated mining process (see Section 3, Step 1) needed to rely on a keyword-based search of files linked in the `README.md`. While the search made it possible to identify a considerable number of views (14k), views that did not explicitly contain the substring "architect" in their filename could not be identified. In addition, albeit used as main source of GitHub repository documentation [21,28], files not linked to `README.md` or `CONTRIBUTING.md` files could not be identified. This choice constituted a tradeoff between internal and external validity, which made it possible to automatically identify views while limiting the number of potential false positives. In future work, this threat could be addressed by using a more sophisticated view identification strategy, *e.g.,* by training an image classification model on the dataset constructed for this research, and automatically classify views from a more extensive pool of images.

Reliability. To ensure the reproducibility and verifiability of our results, we make a comprehensive replication package of the study available online (see Section 8).

7 Conclusions and Future Work

In this research we present an investigation on the architectural views state of practice in open-source projects. The study uses repository mining to build a dataset of 15k views, which are then analyzed both quantitatively and qualitatively.

From our study a clear picture on the use of architectural views in open-source project emerges. Views are used to intuitively convey in an informal manner key architectural concepts, are seldom updated, and are a single-contributor responsibility. A shared syntactic ground between views seems hard to be found, with *ad hoc* viewpoints being a widespread norm. Informally, the sole exception might be the general-purpose architectural set of icons and connectors supported by the AWS Architecture Center[16], that seems to be

[16] `https://aws.amazon.com/architecture/icons`. Accessed 17th April 2024.

shared by a minority of the analyzed views. Overall, it appears as if the effort needed to create views, combined with the potential lack of intuitive yet structured set of viewpoints, hinders utilizing more than one view, and even keeping that single view up to date.

As future work, the topic of views in projects could be further investigated, *e.g.,* by conducting a survey with repository contributors. This would allow to further understand what influences view selection, how relying on a single view affects system understanding and evolution, and what are the current requirements for new viewpoints. Building further on this study, we envision also to conduct research to automatically identify and classify architectural views, and characterize the contributors involved in editing architectural views. Paving the way for future research, our ultimate goal is to define and popularize versionable, templateable views that can be integrated in collaborative programming environments. Given its practical nature, such goal can be achieved solely through empirical evidence and close collaboration with open-source communities, in order to define views promoting simplicity, adoption, and consistency across projects.

8 Data Availability

To support reproducibility and verifiability, we make all data used in this study, scripts, settings, coding guide, and results available in a replication package online[17]. To encourage open science, the replication package is shared under open-source MIT license.

References

1. IEEE Standard for Information Technology–Systems Design–Software Design Descriptions. IEEE STD 1016-2009 pp. 1–35 (2009)
2. International Standard for Software, systems and enterprise Architecture description. ISO/IEC/IEEE 42010:2022(E) pp. 1–74 (2022)
3. Alshuqayran, N., Ali, N., Evans, R.: A systematic mapping study in microservice architecture. In: International Conference on Service-Oriented Computing and Applications. pp. 44–51. IEEE (2016)
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Professional, 4 edn. (2021)
5. Buchgeher, G., Schöberl, S., Geist, V., Dorninger, B., Haindl, P., Weinreich, R.: Using architecture decision records in open source projects–an msr study on github. IEEE Access (2023)
6. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stafford, J.: Documenting Software Architectures: Views and Beyond. Addison-Wesley (2011)
7. Ding, W., Liang, P., Tang, A., Van Vliet, H., Shahin, M.: How do open source communities document software architecture: an exploratory survey. In: André, É., Zhang, L. (eds.) Proceedings - 19th International Conference on Engineering of Complex Computer Systems, ICECCS 2014. pp. 136–145. IEEE, Institute of Electrical and Electronics Engineers (2014)
8. Garlan, D.: Software architecture: a roadmap. In: ICSE '00: Proceedings of the Conference on The Future of Software Engineering. pp. 91–101. Association for Computing Machinery, New York, NY, United States (2000)
9. Geiger, R., Varoquaux, N., Mazel-Cabasse, C., et al.: The Types, Roles, and Practices of Documentation in Data Analytics Open Source Software Libraries. Computer Supported Cooperative Work **27**, 767–802 (2018)
10. Ghanam, Y., Carpendale, S.: A survey paper on software architecture visualization. University of Calgary, Tech. Rep p. 17 (2008)
11. Gousios, G., Vasilescu, B., Serebrenik, A., Zaidman, A.: Lean GHTorrent: GitHub Data on Demand. In: Proceedings of the 11th Working Conference on Mining Software Repositories. p. 384–387. MSR 2014, Association for Computing Machinery, New York, NY, USA (2014)

[17] `https://figshare.com/s/a796b8b414bbc7d09fe2`. Accessed 24th April 2024.

12. Hebig, R., Quang, T.H., Chaudron, M.R.V., Robles, G., Fernandez, M.A.: 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. p. 173–183. MODELS '16, Association for Computing Machinery, New York, NY, USA (2016)
13. ISO/IEC 25010: Systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models (2023)
14. Jenner, B., Flick, U., von Kardoff, E., Steinke, I.: A companion to qualitative research. Sage (2021)
15. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D., Damian, D.: An in-depth study of the promises and perils of mining GitHub. Empirical Software Engineering **21 (5)**, 2035–2071 (2016)
16. Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A.: What industry needs from architectural languages: A survey. IEEE Transactions on Software Engineering **39**(6), 869–891 (2012)
17. Malavolta, I., Lewis, G.A., Schmerl, B., Lago, P., Garlan, D.: Mining guidelines for architecting robotics software. Journal of Systems and Software **178**, 110969 (2021)
18. Muszynski, M., Lugtigheid, S., Castor, F., Brinkkemper, S.: A Study on the Software Architecture Documentation Practices and Maturity in Open-Source Software Development. In: IEEE International Conference on Software Architecture. pp. 47–57 (2022)
19. Ozkaya, M.: What is software architecture to practitioners: A survey. In: International Conference on Model-Driven Engineering and Software Development (MODELSWARD) (2016)
20. Ozkaya, M., Erata, F.: A survey on the practical use of UML for different software architecture viewpoints. Information and Software Technology **121**, 106275 (2020)
21. Prana, G., Treude, C., Thung, F., et al.: Categorizing the Content of GitHub README Files. Emp Software Eng **24**, 1296–1327 (2019)
22. Rost, D., Naab, M., Lima, C., von Flach Garcia Chavez, C.: Software architecture documentation for developers: A survey. In: Software Architecture: 7th European Conference, ECSA 2013, Montpellier, France, July 1-5, 2013. Proceedings 7. pp. 72–88. Springer (2013)
23. Rozanski, N., Woods, E.: Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley Professional (2005)
24. Saldaña, J.: The coding manual for qualitative researchers. Sage (2021)
25. Shahin, M., Liang, P., Babar, M.A.: A systematic review of software architecture visualization techniques. Journal of Systems and Software **94**, 161–185 (2014)
26. Smolander, K.: What is included in software architecture? a case study in three software organizations. In: Proceedings Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems. pp. 131–138 (2002)
27. Tu, Q., Godfrey, M.: The build-time software architecture view. In: Proceedings IEEE International Conference on Software Maintenance. ICSM 2001. pp. 398–407 (2001)
28. Venigalla, A.S.M., Chimalakonda, S.: What's in a github repository?–a software documentation perspective. arXiv preprint arXiv:2102.12727 (2021)
29. Verdecchia, R., Engström, E., Lago, P., Runeson, P., Song, Q.: Threats to validity in software engineering research: A critical reflection. Information and Software Technology **164**, 107329 (2023)
30. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science & Business Media (2012)
31. Zagalsky, A., Feliciano, J., Storey, M.A., Zhao, Y., Wang, W.: The Emergence of GitHub as a Collaborative Platform for Education. In: ACM Conference on Computer Supported Cooperative Work & Social Computing. Association for Computing Machinery (2015)