# Disseminating Architectural Knowledge on Open-Source Projects

## A Case Study of the Book "Architecture of Open-Source Applications"

Martin P. Robillard
School of Computer Science
McGill University
Montréal, QC, Canada
martin@cs.mcgill.ca

Nenad Medvidović
Computer Science Department
University of Southern California
Los Angeles, CA, USA
neno@usc.edu

## ABSTRACT

This paper reports on an interview-based study of 18 authors of different chapters of the two-volume book "Architecture of Open-Source Applications". The main contributions are a synthesis of the process of authoring essay-style documents (ESDs) on software architecture, a series of observations on important factors that influence the content and presentation of architectural knowledge in this documentation form, and a set of recommendations for readers and writers of ESDs on software architecture. We analyzed the influence of three factors in particular: the evolution of a system, the community involvement in the project, and the personal characteristics of the author. This study provides the first systematic investigation of the creation of ESDs on software architecture. The observations we collected have implications for both readers and writers of ESDs, and for architecture documentation in general.

## CCS Concepts

**•Software and its engineering → Software architectures; Documentation;**

## Keywords

Architecture Description, Open-Source Software

## 1. INTRODUCTION

Large software systems realize a number of important design decisions that are intended to remain stable [39, 44, 51]. These decisions and the properties they induce are often referred to as a software system's architecture [41]. Knowledge of a system's architecture helps ensure the preservation of its conceptual integrity in the face of on-going modifications. However, while "every system has an architecture, whether it is documented or not" [41], an architecture must be documented for stakeholders to take it into account.

There have been a number of well-known attempts to codify architectural documentation: from the establishment of standard architectural views as a documentation basis [31, 41], to semi-formal and formal architecture description languages [37, 44], to the standard Unified Modeling Language (UML) [35], to suggested "best practices" for producing architectural documentation [14]. Despite these efforts, however, there is no universally agreed-upon method for documenting a software system's architecture.

In recent years, we have witnessed the growing emergence of informal attempts at communicating software architecture knowledge through essays that combine narrative prose, diagrams, and in some cases source code. This kind of *essay-style documentation* is already publicly available for many systems (e.g., Chromium [8], Firefox [22]), application domains (e.g., consumer electronics [55], mobile robotics [36]), and computing paradigms (e.g., grid [23], cloud [38]). Essay-style documents (ESDs) on software architecture target the broad dissemination of knowledge, as opposed to the systematic specification of the properties and elements of a software system. Their increasing prevalence as vehicles for capturing important software development ideas and experiences makes them extremely relevant to the software architecture discipline. As their popularity and availability grows, we can expect them to play an increasing number of roles in software projects, from recruiting and on-boarding new developers to maintaining architectural conformance.

Without a good understanding of the ESD approach to architectural documentation, we are limited in our ability to properly interpret the information they capture and to help improve their quality. As a first step towards the better understanding of the phenomenon of essay-style documentation of software architecture, we conducted a case study of a book project in which each chapter was the description of the architecture of an open-source software system [10]. Our study relies on multiple surveys and interviews with the authors of 18 of the book's chapters, or 40% of the applicable chapters. To ensure that the data we collected could be interpreted in a meaningful way, we analyzed in detail many other contextual aspects of the case, including by studying the text of all chapters and by interviewing the editors of the book.

The major contributions of this paper include *(1)* a synthesis of the process of authoring ESDs on software architecture; *(2)* a series of observations on three important factors that influence the content and presentation of architectural

*ICSE '16, May 14-22, 2016, Austin, TX, USA*
© 2016 ACM. ISBN 978-1-4503-3900-1/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2884781.2884792

knowledge in this documentation form; *(3)* a set of specific recommendations for readers and writers of essay-style documents on software architecture.

The three influential factors we surfaced are the evolution of a system, the community involvement in the project, and the personal characteristics of the author. In addition to the inevitable fact that software architecture is a function of the technical aspects of a system, the insights that emerge from the study motivate the formulation of a hypothesis we call the *multiple lens hypothesis*, which states that architectural information is focused from four different types of lenses: *(1)* technical, *(2)* historical, *(3)* personal, and *(4)* social. Although non-technical elements have been part of software architecture descriptions for decades, our study surfaces new and unexpected insights about the relation between technical factors and the other lenses. For example, we describe how some architects used stories of the evolution of their system to describe and justify design rationale, as opposed to the traditional approach of capturing rationale to support evolution.

This paper is organized as follows. Section 2 details the design of our case study and Section 3 the context for the case under study. Section 4 summarizes our key findings, while Section 5 discusses their implications, threats, and limitations. Section 6 describes the related work that informed the study and provided its theoretical departure point. Section 7 concludes the paper.

## 2. CASE STUDY DESIGN

There are so many different contexts in which open-source contributors produce architectural documentation that attempting to study the phenomenon generally bears the risk of becoming intractable. To scope our initial investigation of the phenomenon, we decided to conduct a *case study* of the production of architectural descriptions for the book "Architecture of Open-Source Applications" [10], henceforth referred to with its acronym AOSA.

### The Case

As part of the AOSA book project, co-editors Amy Brown and Greg Wilson recruited 74 open-source contributors to author a total of 49 chapters, of which 45 described the architecture of different open-source systems; four of the chapters are not strictly on the architecture of a system, but address more general topics (e.g., release engineering).[1] The project started in the spring of 2010 and was completed with the publication of the second volume in June 2012. The authors were instructed to write about 10 pages on the major design decisions for their system, and to write for an audience of experienced software developers. Given the relatively loose initial editorial guidance, the authors produced a body of work that varies greatly in quality, style, and choice of topics. The result of this effort is freely accessible [10].

Our research design is thus that of the *embedded multiple case study* [58] with two levels of context. The book project as a whole provides a first, general, context (recruitment strategy, editorial process, review process, etc.), and the creation of each chapter forms a distinct embedded case with its own context (type of system, technology, development process, etc.). We distinguish between these two levels of context with the terms *general context* vs. *system context.*

[1]These are chapters 9 and 13 in Vol.1, and 1 and 2 in Vol.2.

**Table 1: Study Participants. *ID*: an alias for the participant; *Ch*: chapter reference (*volume-chapter*); *System*: the type of project; *Occ.*: participant's occupation; *Role*: the participant's role in the project.**

| ID | Ch. | System | Occ. | Role |
|---|---|---|---|---|
| 1 | 1-2 | Audio editor | Technical | Developer |
| 2 | 1-6 | Development env. | Technical | Contributor |
| 3 | 1-10 | Communication | Technical | Developer |
| 4 | 1-12 | Version control | Technical | Developer |
| 5 | 1-19 | Web application | Technical | Developer |
| 6 | 1-24 | Image processing | Technical | Founder |
| 7 | 2-3 | Operating system | Technical | Outsider |
| 8 | 2-5 | Compiler | Research | Founder |
| 9 | 2-9 | Image processing | Technical | Developer |
| 10 | 2-11 | Data visualization | Technical | Developer |
| 11 | 2-12 | Web content platform | Media | Contributor |
| 12 | 2-13 | Web content platform | Technical | Developer |
| 13 | 2-14 | Web server | Business | Founder |
| 14 | 2-15 | Middleware | Technical | Founder |
| 15 | 2-20 | Persistence framework | Technical | Founder |
| 16 | 2-22 | Web app. framework | Technical | Founder |
| 17 | 2-23 | Build tool | Technical | Developer |
| 18 | 2-24 | Middleware | Technical | Founder |

### Research Questions

Our case study seeks to answer the following questions:

**RQ1:** How do authors approach the production of essay-style documentation of software architecture?

We define *essay-style documentation (ESD)* as software architecture description in free narrative form, as opposed to architectural documentation that follows prescribed, systematic templates [14] or standards [29]. In addition to book chapters [10, 45, 53], formats for ESD on software architecture include technical reports (e.g., on the Chrome Browser [8]) and on-line articles (e.g., on the Firefox OS [22]).

Given a set of ESDs and a proper description of the context in which they were produced, we then seek to answer:

**RQ2:** What factors influence what authors select to include in an essay-style informal document on software architecture, and how they present it?

### Case Selection

This case study relies on interview data from the editors and authors of the AOSA book. As in most case studies, we had to follow an opportunistic approach to participant recruitment: We invited every author to participate and involved all volunteers in the research (up to one per chapter). A total of 18 authors of 18 distinct chapters participated in the study.

Table 1 summarizes relevant attributes of the study participants, which map one-to-one with cases. Although participants agreed to be personally identified, we refer to them through identifiers to afford them a certain degree of privacy: $P_x$ refers to Participant number $x$. The table includes selected contextual information. We include occupation to distinguish between participants with a primarily technical occupation (software engineering and variants) from a researcher, media expert, and business person. The information about the participant's role in the project is our categorization based on the participant's self-reported qualification. The roles have an implied order of importance. With *Founder*, we indicate a person who was critical to the inception of the project and (except for $P_5$), acted as the main architect or co-architect of the system. *Developer* indicates a person who makes substantial technical contributions to the project and exercises some degree of leadership or authority in the project. *Contributor* indicates someone who may have contributed to the project in non-developer capacity. *Outsider* is someone who is not involved in the technical aspects of the project as a contributor, but who has technical knowledge of the system. This variety of roles is reflective of how architectures are documented in more traditional development settings, where, in practice, different stakeholders may author or contribute to such documents.

### *Data Collection*

We began with a close reading of a sample of chapters to get acquainted with the material and identify salient aspects (e.g., use of source code, informal diagrams). In early 2014, we asked one of the book's editors (Wilson) to invite (by email) all 74 of the authors to participate in this project. Authors of 14 different chapters agreed to participate (Phase I of the data collection).[2] We sent each of the 14 authors a questionnaire, shown in Figure 1, comprising seven questions about the context for their contribution and their approach for writing the chapter. Upon receiving a completed questionnaire, we scheduled a follow-up interview with each author. The interviews were semi-structured: we prepared an interview guide by studying each participant's chapter and questionnaire responses and adapting a baseline interview template to the specifics of author and their chapter [43, §2.2]. For example, for the participant who was a project outsider, we used the interview to learn about the participant's connection to the project, a piece of information that was not required from other participants. The interviews, which lasted around 60 minutes on average, were conducted using text-messaging software so that we could collect the exact transcript of each interview. We also interviewed the two co-editors of the book to collect information on the general context of the project.

Following the analysis of the questionnaires and interviews, we emailed each participant a validation questionnaire. In May 2015 we sent to the authors of the remaining chapters a follow-up invitation to contribute to the project. Four more authors submitted the questionnaire and participated in an interview (Phase II of the data collection). We did not send the validation questionnaire to the Phase II participants because their interviews focused on themes already validated as part of Phase I. Finally, we sent each participant a draft of this paper and asked them to flag any errors, and to comment on the credibility and usefulness of the findings: 13 participants responded. In the end, our case study can take into account the experience of 20 authors and editors in the writing of 18 chapters, or exactly 40% of the 45 applicable chapters.

### *Data Analysis Method*

We analyzed the questionnaire responses and interview transcripts using a qualitative approach [43] adapted from grounded theory methods [16].

The initial analysis of the chapters was exploratory, and served as source of input for designing the survey and interview instruments. We then systematically analyzed the questionnaire responses and interview transcripts using open

[2]In several instances, a single author of a multi-author chapter replied to our invitation but also copied their co-authors.

1. How do you describe your role in the project described in the chapter? If there were other architects, what was the division of responsibilities? [**Role**]
2. Did you consult any other people in writing the chapter? If so, who were they? What was the purpose of consulting them? [**Collaborators**]
3. What were the major types of topics (a.k.a concerns, aspects, or issues) of the system that you decided to describe in your chapter. Why did you select those? What did you leave out? [**Main Topics**]
4. What approach did you follow for presenting different topics? This includes the choice of figures and code snippets and their notation, and the writing style you adopted. [**Writing Approach**]
5. How did you decide on the sequence of topics (what to present first, second, etc.)? Is this sequence important to understand the system? [**Topic Sequence**]
6. Did you use any materials (documents, diagrams) in writing the chapter? If so, what are those materials? [**Supporting Material**]
7. Did the fact that the chapter was intended for a broad audience impact how you approached/wrote it? If so, in what ways? Did you include, change, or omit information because this was a book chapter and not an architectural design document? If so, what information? [**Impact of Broad Audience**]

**Figure 1: Pre-interview questionnaire. The summary keywords following each question are for ease of reference in the paper: they did not appear on the questionnaire.**

coding: we assigned codes to sentences or paragraphs and we defined the codes as the study progressed. We subsequently used axial coding and went through the codes to link them to categories of code that synthesize high-level concepts (e.g., *use of diagrams*). We jointly completed this analysis while immersed in the data during a week-long work session during which we were physically collocated.

As a result of this initial analysis, we identified three important themes (evolution, community, and personal factors) and reached out to additional participants to specifically develop them. At the end of Phase II, in which we interviewed four additional participants, we systematically re-coded all transcripts and questionnaires using the three codes: evolution, community, and personal factors. We then re-analyzed the data along each dimension, and produced the interpretations described in Section 4.2 (paragraph "Main Themes"). We also integrated the input of the participants in our final version.

## 3. GENERAL CONTEXT

In this section, we distill the critical aspects of the general context in which architectural descriptions were produced.

### Editorial Process

This section is a summary of the editorial process based on the interview with the book editors.

The project proceeded in two phases corresponding to the two volumes. The recruitment process involved email advertisement and solicitation, and snowball recommendations ("friend-of-a-friend introductions"$_{\text{Wilson}}$). The authors worked on their chapters asynchronously and at different paces, which varied from a few days to several months.

The text of an email for the authors included the following guidelines: *(a)* "Our target is approximately 10 pages per chapter, but can go higher..."; *(b)* "Your audience [has] probably been programming for five years or more on top of a [B.Sc.] in some computer related discipline [...] but hasn't worked in your specific area [...]"; *(c)* "Somebody once defined software architecture as the set of decisions made in writing an application. I think that's a good rule for deciding what should be in or out of your chapter: what are the things that are only "obvious" in retrospect? [...] (Another way to think about the chapter is what the whiteboard talk you'd give to a new developer who was about to start coding for you.)".

Some authors did not strictly adhere to the guidelines, a development that was recognized and accepted by the editors "if authors thought the most important/useful thing was to say "XYZ", who was I to tell them they were wrong?"$_{\text{Wilson}}$, except for "holding the line on [...] marketing"$_{\text{Brown}}$.

Each chapter was sent out to two reviewers. The reviews were returned to the editors, who forwarded the appropriate comments to the authors with a request to incorporate them into the final version. The chapters were then lightly copy-edited by the two book editors. Although the editors gave feedback to the authors, the authors' creative license was extensive, and "what you get in each chapter is often exactly what the author originally wrote"$_{\text{Brown}}$. The editors also did not give guidance about diagrams, except that they wanted many. They did not say anything about the use of source code.

## Chapter Metrics

Table 2 provides baseline metrics on the chapters that were the target of our case study. The chapters are identified in the form *volume-chapter*. The size metric is the number of words of text excluding code blocks, punctuation, and tables. The number of system diagrams corresponds to the number of diagrams that describe the structure of the system. We excluded from this count the figures that depict screen-shots of an application or the application's output. In the column listing the number of diagrams, we also indicate in parentheses the number of system diagrams that were drawn in the Unified Modeling Notation (UML) as a subset of the total. Finally, the last column indicates the number of code blocks in the chapter. The median values in the last row of the table are the median values across all 45 applicable chapters. The complete metrics for the remaining chapters are omitted for lack of space.

## Collaboration Process

Although all authors interacted with the editors and reviewers of their chapter, we noticed different collaboration models for obtaining the information for a given chapter from other stakeholders. The authors engaged in three types of collaboration processes: *solo effort*, with *primary informants*, or through *community feedback*.

*Solo Effort.* Nine participants clearly indicated that they authored their respective chapters alone ($P_{1,3,12,14,15,16,18}$) or only with their co-author ($P_{6,8}$). In all cases these authors were developers or founders of their project (see Table 1).

*Primary Informants.* In five instances, the participant authored their chapter with the help of one or two principal informants ($P_{4,7,9,10,17}$). In all cases, the primary informants were the project's founder or core developers. The contributions of the primary informants included

- answering questions "I might have asked a question or two [to] the primary architect"$_{P4}$;
- validating the architecture description "[We] had [X and Y] (both major contributors of the life of the project) read early drafts of the chapter for clarity, accuracy and completeness"$_{P10}$;
- providing feedback on content selection "The purpose of consulting them was to get feedback on the chapter outline, and the amount of focus put on each one of the sections"$_{P9}$.

*Community Feedback.* The four remaining participants relied on community feedback when putting together their respective chapters. $P_2$ and $P_5$ were technical contributors to the system described who interviewed the other major contributors for questions and feedback. $P_{11}$ was a non-technical contributor to the project who set up a community feedback process through a Wiki "we consulted dozens of people to get information and make sure our understanding was correct."$_{P11}$ $P_{13}$ was a founder of the project who was not involved in technical work, and who relied on input from developers to author the chapter. "I consulted [...] the original author of [the system] (and the principal co-founder of the company), as well as the other core developers we employ. I also ran the drafts against a handful of the most prominent 3rd party developers... X in particular gave a permission to re-use quite a bit of his guide [...] - that one became the foundation for the last section."$_{P13}$

## Supporting Material

Most authors used a combination of information sources to produce their chapter, including the application source code,

**Table 2: Chapter metrics. *Chapter*: the chapter ID in the form *volume-chapter*; *Nb. Au.*: the number of authors; *Size*: The number of words of text; *Dia.*: The number of system diagrams, with the number of diagrams drawn using UML in parentheses; *Code*: the number of code fragments. Note that the *Median* values are computed for all 45 relevant chapters of the AOSA book.**

| Chapter | Nb. Au. | Size | Dia. (in UML) | Code |
|---|---|---|---|---|
| 1-02 | 1 | 6236 | 4 (0) | 1 |
| 1-06 | 1 | 7360 | 9 (0) | 11 |
| 1-10 | 1 | 4987 | 4 (0) | 14 |
| 1-12 | 1 | 6416 | 3 (0) | 2 |
| 1-19 | 1 | 4169 | 10 (0) | 11 |
| 1-24 | 2 | 5780 | 4 (0) | 11 |
| 2-03 | 1 | 4375 | 4 (0) | 12 |
| 2-05 | 2 | 10 736 | 1 (0) | 8 |
| 2-09 | 2 | 8985 | 7 (0) | 8 |
| 2-11 | 2 | 5217 | 2 (0) | 7 |
| 2-12 | 2 | 7767 | 1 (0) | 0 |
| 2-13 | 1 | 5968 | 1 (0) | 22 |
| 2-14 | 1 | 7041 | 1 (0) | 0 |
| 2-15 | 1 | 5695 | 3 (0) | 3 |
| 2-20 | 1 | 8197 | 16 (7) | 13 |
| 2-22 | 1 | 6638 | 1 (0) | 15 |
| 2-23 | 1 | 4800 | 3 (0) | 7 |
| 2-24 | 1 | 6125 | 8 (0) | 2 |
| **Median** | 1 | 6400 | 3 (0) | 5 |

the project web site, a developer Wiki, mailing lists, bug descriptions, etc. Although most chapters do not provide an explicit list of references, chapter 2-10 (GNU Mailman [56]) is an exception that lists the typical collection of information sources available to authors. Although it is not possible to reliably estimate to what extent each author used supporting material of each type as part of their work, the answers to Question 6 (Supporting Material) in the pre-interview questionnaire (Figure 1) confirmed that at least seven authors relied on existing technical documentation on the respective projects' websites. In addition to technical documentation, one participant ($P_9$) mentioned that he ran some code metrics tool, and two participants ($P_{14,15}$) mentioned that they reused diagrams they had drawn previously for some other purpose: "Some of the figures I used were influenced by diagrams I had presented in talks in the past"$_{P14}$

## 4. RESULTS

We organize our main observations in terms of the two research questions (see Section 2). Section 4.1 explains how authors approached the production of essay-style documentation (ESD) of software architecture (RQ1). Section 4.2 elaborates on the factors that influenced what authors selected to include in an ESD and how they presented it (RQ2). In Section 5 we build on these observations to discuss the lessons we can take away from the study and the implications for the documentation of software architecture.

### 4.1 Documentation Perspective

To understand *how authors approached the production of essay-style informal software architecture descriptions* (RQ1), we first clarify who they had in mind as an **audience**. This leads us to analyze the question of the use of **source code** and **diagrams** from the perspective of reader accessibility. With these elements in place, we then shed light on the other **provisions the authors made to adapt the technical content** to their audience.

#### *Audience*

The review of the chapters and the results of the questionnaires and interviews confirm that the authors respected the editorial guidelines about the target audience: the contents were aimed at a "general programming audience"$_{P1}$. At least 13 participants had made explicit mentions to this effect, such as "I think I tried to target somewhat experienced software developers that didn't have any familiarity with [the system] or its codebase"$_{P4}$.

As far as knowledge of the domain is concerned, there was more variety: Some authors assumed no specialized domain knowledge "I included more overview text than if my audience were already familiar with embedded systems"$_{P7}$. Only three authors mentioned that they assumed knowledge of some related technology or concepts specific to their chapters. In response to the question "to what extent did you feel you could rely on the general familiarity of your audience with Web servers..." $P_{13}$ replied "I presumed "to a great extent"". The two other similar cases we encountered were $P_{15}$ who expected "...some degree of Python, and some degree of relational database knowledge" and $P_{16}$ who commented that his target audience "was likely to be familiar with either MVC[3] or Haskell, possibly both".

[3]The Model-View-Controller architectural pattern.

#### *Use of Source Code*

As Table 2 shows, almost all chapters contain code fragments. The tacit assumption made by authors is that readers will be able to understand code. However, only four authors mentioned this requirement explicitly: $P_{15}$ and $P_{16}$, quoted above, and $P_{12}$ "I felt I had to assume PHP knowledge". The developers of a compiler for Haskell (itself written in Haskell), "deliberately tried to write for an audience that was not deeply familiar with Haskell"$_{P8}$, a thought that was almost identically echoed by $P_{16}$ "I tried to provide code snippets that users not familiar with Haskell would be able to understand/appreciate".

While prior research on architectural description [13, 14, 33, 37, 51] has been largely divorced from implementation-language concerns, such concerns figured prominently in the ESDs we studied. For example, the four authors mentioned above expressed sensitivity to this issue in cases of languages with smaller user bases. This was especially noticeable when contrasted with the widespread disregard for the same issue in systems written in popular languages (Java, C, C++). This point was summarized well by $P_{18}$, who purposely did not use any code fragments[4] "...one thing that often gets in the way is the language; if I included examples in C, I would have excluded Pythonistas, Ruby programmers, etc., from the audience."$_{P18}$.

#### *Diagram Notation*

Most chapters had at least one diagram (see Table 2), but only a small fraction made use of UML or another existing language. Under 10% of diagrams across all 45 chapters (14/164) were drawn in UML, and no chapter author predominantly relied on UML. Furthermore, no diagram provided an explicit legend for the employed notation. Interestingly, no responses to Question 7 (Impact of Broad Audience) of the questionnaire from Figure 1 mentioned diagrams. We probed this question during the interviews, and even then only three authors explicitly commented on the link between diagrams and the audience. Two explicated their assumption about the lack of knowledge of a specific notation: "So when doodling a diagram I make sure I don't expect people to know some conventions in order to understand it..."$_{P3}$; "In my opinion most readers are not well versed in UML"$_{P11}$. The third author pointed out the informal nature of the document "this is not technical documentation, mind you!"$_{P18}$. The conclusion we draw from this evidence is that the precision and rigor of the modeling notation were not a priority for the participants.

#### *Adaptation for the ESD Format*

Although the general context is clearly the writing of an essay as opposed to systematic software documentation, the authors were nevertheless tasked with communicating knowledge on the architecture of their system. We investigated how they bridged this dichotomy by systematically analyzing the responses to Question 7 in the questionnaire (Impact of Broad Audience), as well as related fragments from the interviews.

Given that the length of the chapters was restricted, we took into account that the scope of the treatment was limited and that authors had to make choices about what to include "I did not distort any information. Just limited the scope of what I wrote about."$_{P12}$ The extent of the adaptations made for the essay style varied substantially between authors. Four

[4]The only exception were two instances of client-side usage of the system's API.

authors indicated they made few concessions to the essay style ($P_{4,5,7,12}$), e.g., "I wouldn't do much differently if this were an arch design doc"$_{P7}$. In contrast, $P_8$ focused heavily on what a general reader "would find interesting". Given this spectrum, we focused our analysis on two themes: the addition of *background* (discussed by $P_{2,5,7,9,13,14,15,17}$) and the *granularity* of the descriptions (discussed by $P_{1,3,6,8,10,11,16,18}$).

In our analysis, the term **background** refers to general domain knowledge and historical facts that would be known to the system's developers. The addition of background information to a chapter was a common device to reach out to a broader audience, but also to bridge perceived gaps in technical knowledge. For example, "I included more overview text than if my audience were already familiar with embedded systems"$_{P7}$; "I explained the background because the problem area [...] is not well-known and common..."$_{P14}$. These comments show how the authors engaged in a reflection on their target audience and the background that is necessary to grasp the architecture of their system. Background information was also provided to give the reader a historical perspective which, in some cases, was intended to illuminate the *rationale* for some architectural decisions. "An architectural design document would have been very dry and artificial, since it would have described the engineering features [...], without explaining why they came to be"$_{P9}$.

The question of **granularity** refers to the manner in which the authors tackled the challenge of abstracting details of their system in a way that is compatible with the essay-style documentation. ESD is not amenable to more dynamic documentation features such as collapsible tables, cross-reference tools, and interactive visualizations. On the one hand, in this context the authors talked about avoiding "internals"$_{P3}$, "omitting details"$_{P8}$, that "change over time"$_{P10}$, not delving into "technicalities"$_{P18}$, or presenting only the "core essence"$_{P6}$. On the other hand, we found it much more difficult to elicit specific, concrete statements about how the abstractions were tailored to the chapter format.

## 4.2 Content Selection and Presentation

To understand the factors that guided the authors in selecting the specific content they presented in their respective chapters, we first analyze the answers to the three relevant questions from the pre-interview questionnaire (Figure 1). We then elaborate on the three principal themes identified in our analysis: (1) importance of a *system's evolution* in capturing its architecture, (2) central role of the system's *development community*, and (3) *personal background and preferences* of the architecture document's author.

### *Questionnaire Answers*

Questions 3–5 (Main Topics, Writing Approach, Topic Sequence) targeted directly the issue of content selection and presentation. The answers to these questions served as the seed for the interviews.

*Question 3: Main Topics.* Five participants ($P_{3,6,8,14,18}$) simply indicated that their focus was to present "the general structure of the project"$_{P3}$, "a broad overview"$_{P6}$, or some similar generality. This was not a given however: $P_{10}$ went in the other direction "We talked mainly about the internals of [the system]". Six authors answered by listing one or a few system features or other technical aspects, such as modularity ($P_2$), data structures ($P_{4,7}$), persistence issues or components ($P_{12,17}$), and parallelism ($P_9$). In addition, $P_9$ also indicated that he chose to put "a lot of emphasis on the fact that the software architecture reflects the composition of the developer community, and their needs". Five authors indicated that their topics focused on providing a historical perspective that served to explain the requirements of the system ($P_{5,11,13,16,18}$). $P_5$'s answer provides the best summary for this kind of focus: "I focused on a chronological narrative [...]. These [historical developments] provide the context of requirements on the system architecture". Finally, two authors ($P_{1,15}$) provided extended, in-depth justifications of the topics of their chapter, which we leveraged in our analysis of the main themes (below).

*Question 4: Writing Approach.* This question may have required too much a posteriori introspection and many authors found it, rightly, challenging. Four answered they did not know ($P_{2,8,11,13}$), seven ($P_{4,6,7,9,10,16,18}$) provided superficial statements ("informal tone"$_{P4}$, "casual writing style"$_{P7}$), one related the writing approach to the expected audience ($P_{17}$), and one focused on the linguistic aspects ($P_5$). Two authors answered that they approached the writing by "explaining this to someone like myself"$_{P3}$: "I thought about what I would want to read, if approaching the project fresh, and the questions I would have"$_{P1}$. $P_{15}$ described a variant of this idea "The code snippets and figures were largely based on things I've explained in other formats for years...". Two answers were particularly original in our context: $P_{12}$ structured the chapter "around a minimal working [...] script", and $P_{14}$ emphasized the narrative approach "In general, I like to tell stories to convey information".

*Question 5: Topic Sequence.* The dominant answers for this question were variants of the bottom-up (i.e., details first) ($P_{4,10,16}$), and top-down (i.e., details last) ($P_{1,5,6}$) orders of presentation. $P_8$ was not concerned with sequence "We just identified the big topics [...] and wrote a section on each". $P_{2,3}$ were influenced by the chronological development of the system: "I subconsciously just follow the history of the project and go about presenting things in the order they were built"$_{P3}$. $P_7$ presented the material in "descending order of importance" and $P_{18}$ in increasing order of complexity: "I've tried to present topics more accessible to general public first..."$_{P18}$. $P_{15}$ chose one of only two perceived options given the architecture of the system "either learn the Core first from the inside out and then the ORM, or learn the ORM first and then dive into the underlying Core structures"$_{P15}$. The order of presentation for $P_{12}$ naturally follows from his decision to present the construction of a script, and the same applied to $P_{14}$, albeit to a lesser extent: "I think the sequence I presented fit nicely within the "story" concept". $P_{17}$, who was describing the architecture of a build system, followed the order of the build pipeline. The remaining authors ($P_{9,11,13}$) did not provide an answer that we could clearly interpret.

### *Main Themes*

In their comments, our participants provided numerous insights on how technical and domain aspects of the project influenced the content and presentation of their essay. For example: "The main aspects discussed were the Data Pipeline, Parallelism,... They are at the core of the needs of image processing..."$_{P9}$. This was, however, largely expected, and documentation of the technical aspects of architectural descriptions is treated in numerous other sources (e.g., [14, 34, 35, 37, 44, 51]).

The major original elements we surfaced as part of our analysis concern the importance of the **system's evolu-**

**tion**, its **development community**, and the **personal characteristics and preferences** of the author. Given the space available we focus our report on the insights we gathered related to these three themes.

*Evolution.* Software evolution has been studied from an architectural perspective for a few decades already [39, 44]. Several notations for capturing architectural evolution have been proposed [37], and a number of evolution-inspired patterns and styles have been developed, refined, and put into widespread use [51]. The principles of a system's evolution have also been recognized as a key part of at least one standard definition of software architecture [41, p.12]. However, our study uncovered another role of evolution that is not as readily recognized in the software architecture literature:

> A system's evolution was an active driver of architectural description, rather than a passive system trait to be captured and managed by notations, patterns, and styles.

The following quotes are illustrative: "History can inform your understanding of the structure"$_{P1}$; "recognizing that the architecture is alive is critical to understanding the very nature of software"$_{P9}$.

As we discussed in Section 4.1, authors included information about the history and evolution of their systems to provide background for a general audience. However, we found that in many cases information on the evolution of the system also served to complement and strengthen the technical discussion. Many authors "thought it was important to describe how the architecture had progressed as opposed to presenting it fully formed"$_{P2}$. The reasons are that evolution information illustrates the rationale for architectural decisions "Why the library abstractions forced us to copy data"$_{P1}$, more generally helps to explain the current structure "it's also useful to keep that kind of evolution in mind because you sometimes get that parts of the architecture have been converted to a new style..."$_{P4}$, or actually serves as a way to validate the existing architecture by showing how it enabled certain anticipated evolution paths ($P_6$). $P_2$ and $P_5$ in particular wrote chapters with a strong historical perspective, following a model we would call the "incremental requirements reveal", where each section addresses new features and how they added to the previous version. This model was also followed, to a lesser extent, by $P_4$: "I [...] go about presenting things in the order they were built". A final insight is that, as $P_9$ points out, a system's architecture is fluid, and recent changes may influence the content selection approach of the author: "another key formative influence on the chapter is that it was written shortly after Moodle 2.0..."$_{P12}$.

The information regarding a system's evolution was seen as helpful in elaborating the architecture by most, but not all of the participants. A notable exception was $P_8$, who indicated that he did not see evolution as a relevant factor.

*Community.* Traditional software architectural design is a collaborative process that involves a team of architects whose target audience comprises the project's managers as well as the resulting system's developers, customers, and users [31, 39, 44, 51]. Architectural documents are typically targeted at a subset of these stakeholders. For example, an architect may use the documents to discuss certain design decisions with other architects, or to communicate a vision of the system to developers. In other words, the architecture document is intended *for* this relatively small community.

What we found in the architectural ESDs we studied is very different. "What is missing from many people's picture of open source software development is that it is a technical AND SOCIAL enterprise"$_{P1}$. A major aspect of the general context for this study is the open-source nature of the projects. Eight authors made specific remarks about the impact of the community on the architecture.

> When describing the architecture of an open-source system, the resulting document is *for* but also *about* the contributor community.

This insight is illustrated by the following quotes: "I did not want to dis the project too much"$_{P1}$; "fourth [paragraph] was my attempt to acknowledge the work of 3rd party guys"$_{P13}$; "Communicating WHAT the technical debt is, is necessary for building a consensus"$_{P1}$; "I think one of the first things that I'd like to happen is for a person (hopefully a future contributor) to decide which part of the project they like the most"$_{P3}$.

For a number of projects, the community had a direct impact on the architecture: "the architecture is shaped by social forces as well as technical ones"$_{P1}$; "the community was also very vocal about certain changes"$_{P2}$; "OMPI represents a community, and it's not just based on any one persons opinion"$_{P14}$. In some instances, the developer community influenced the ideas underlying the architecture's design as well as its documentation ($P_{11,12}$). "Lots of people wrote [the wiki page]. I don't know who the main contributor was."$_{P12}$ An important link to the context is that in the projects of $P_{2,6,14}$ a plug-in infrastructure, which allowed community contributions and/or code sharing, was an important part of those systems' architectures.

*Personal factors.* A writing process is necessarily a personal endeavor. The study allowed us to explore how personal characteristics and preferences of authors impacted their architectural descriptions in the context of ESDs. First, as pointed out above, some authors envisioned the information needs of the reader as similar to those they had when they started the project ($P_{1,3,9,14}$), e.g., "The reader that I had in mind was "myself when we were starting the toolkit""$_{P9}$.

In a traditional architectural design setting, it is expected that an expert—likely a system's architect—will document the architecture. By contrast, we learned that the idea of a "monopoly" on architectural knowledge crumbles in many open-source development contexts.

> In the absence of a formal documented architecture, authors shaped an understanding of the architecture from personal experience with and exploration of the system.

In that sense, each resulting ESD reflects its authors' backgrounds, interests, preferences, and knowledge about the system. This phenomenon was particularly pronounced in the case of $P_7$, a project outsider who undertook the task of understanding the architecture specifically for the purpose of writing the chapter: "I created figures in the chapter that I had created on paper while learning about [the system]". While $P_7$ was the only "outsider" among our study participants, this is not an uncommon occurrence: in another recent on-line book effort that provided architectural ESDs of open-source systems, none of the ten chapter authors were the respective systems' contributors [53]. It is important to note that the case of an outsider having to capture a system's architecture is also relevant to the more traditional setting, when architectural documentation falls out of date and must be recovered from the implementation artifacts [39, 51].

Given the restricted size of the chapters, content selection was particularly important. In a traditional setting, the stated or assumed goal of an architectural document is to completely and accurately capture the architecture from one or more perspectives [14, 31, 35, 37] (although it is acknowledged that this is difficult to do in practice [21, 44, 51]). This is not the case with essay-style documents. Many of the authors ($P_{1,2,4,7,12,14,17}$) were candid about the fact that they selected topics based on their personal knowledge or interest for certain aspects of a system. For example, $P_1$ discussed performance at length in the chapter because it was "a personal concern for me [...], not necessarily because it is a key aspect to the success of [the system]." As another example, $P_{12}$ included "the most interesting aspects I had noticed while working on [the system]". In a similar vein, close to one half of the chapter written by $P_{14}$ focuses on the plug-in system of the application, a decision which also originates in the author's prior work: "using plugins for HPC architectures was the topic of my Ph.D. dissertation". $P_{17}$ exemplifies the case of interest-based selection: "[I focused on a certain component] mainly because I'm interested in dependency chains and optimisation and the code around it was interesting".

In addition to topic selection, in an architectural ESD the overall presentation style will also be a reflection of the author's personal characteristics ($P_{2,5,6,9,12,14,15,18}$). Examples included the use of stories ($P_1$), reliance on a specific pattern of interleaving decisions with diagrams ($P_5$), and the decision whether to include references to code ($P_6$) or not ($P_{18}$).

Finally, one of the most consistent observations about the influence of the authors' personal characteristics on architectural descriptions concerns the lack of use of a standard notation—specifically, UML—for diagrams. We found that, in many cases, this could simply be explained by the lack of familiarity of an author with UML or with diagramming tools ($P_{5,6,8,10,13,16,17}$). In most other cases, the authors thought the notation was unfit for the purpose of capturing their vision of the system's architecture ($P_{1,2,3,4,9,14}$), e.g., "there's just no sense in trying to adhere to some part of UML when all I want to do is draw a picture to illustrate an idea I'm trying to convey"$_{P4}$. $P_{18}$ used UML in an "informal way", and $P_{15}$ made a more systematic use of it, corroborating our hypothesis that personal background (in this case, proficiency) determines the inclination to use it: "I worked in UML-ish places in the 90s".

## 5. DISCUSSION

In this section, we reflect upon the implications of our study, describe its quality and credibility attributes, and discuss its limitations.

### 5.1 Implications

This study provides the first systematic investigation of the creation of ESDs for software architecture. The observations we collected have implications for both readers and writers of ESDs, and for the broader field of software architecture documentation.

The major outcome for readers of software architecture ESDs is to facilitate a deeper and more critical interpretation of these documents. Despite the largely uniform process described in Section 3, the process descriptions we provide in Sections 3 and 4.1 illustrate the wide variety of perspectives with which authors can approach how they convey architectural information—including through a tutorial ($P_{12}$), the use of stories ($P_{1,14}$), the relating of historical information ($P_{2,5}$), etc. Before we undertook this project, our view of the AOSA chapters was somewhat unidimensional: we considered that the chapters presented *the* prevalent architectural perspective of a project within a community. We could, of course, note the differences in style and surmise that various factors would influence what we read, but our ignorance of the underlying factors stifled further consideration. The observations we report in Section 4 provide additional texture to the ESDs of the AOSA book, with broader implications on architecture description in general.

The study surfaced how three important non-technical aspects—evolution, community, and author characteristics—are intimately tied to architectural descriptions in a way that was overlooked by previous software architecture work. The study therefore provides a tool for the critical interpretation of ESDs on software architectures. The set of insights that can be derived from it is open-ended, but should include a number of questions about the genesis of the document: What is the role of the author? What part of the application did they work on? How is community involved in architectural decisions? What were the major versions and how did they impact the system? For example, three major insights we personally took away were: *(1)* an ESD tends to become a description of a composite of multiple versions of a system (evolution); *(2)* some architectural information may not be traceable to any particular contributor (community); and *(3)* an author becomes a *de facto* authority on a system by virtue of capturing their understanding of it in the ESD (personal factors).

For writers of ESDs, several observations can serve as the basis for a checklist of important points to consider:

- In contrast to formal architectural documents for which specific stakeholders profiles should be identified [41], the audience for ESDs will typically be open-ended and not well defined. Some of our participants struggled with the question of the accessibility of their material (e.g., how much background to include or whether to use source code). This suggests that, rather than linking the ESD to specific stakeholders, it may be more useful to specify the software development topics that are explicitly captured in the document (e.g., using the ACM Computing Classification System [4]).
- Given the concerns the authors shared about the accessibility of code fragments, if used, source code should be vetted for readability. To a certain degree, this may be possible to support automatically [12].
- The impact of an author's experience and training on the architecture document must be made transparent. To this end, the ESD should include a biography that summarizes the author's involvement with the system in terms that can be related to the ESD's content.
- The sequence of topics presented in the ESD can be planned according to one of the strategies we reported in Section 4.2–*Question 5: Topic Sequence.*
- One should explicitly assign a purpose to those stories of the system's evolution that are deemed architecturally relevant. For example, that purpose may be to document rationale, or to validate key design decisions.

We argue that the implications of this study go beyond the ESD format by providing several insights on the nature of architectural information. *(1)* A system's architecture is inextricably tied to its evolution, both through the role evolution plays in clarifying the design rationale, and in changing (and, in the process, often "breaking") the architecture. In that sense, the evolution is not only an object of architectural design (something that has been previously acknowledged in the literature [44, 51]), but also an active driver of architecture and architectural description. *(2)* An architecture is, obviously, a product of its community, whether small and tightly-knit, large and porous, or anything in between. However, in addition to this common ground, we observed how the architecture can also be a *reflection* of its community in ways that may go beyond Conway's Law [15]. The community-induced forces we learned about may not be unique to open-source projects or ESDs, and we see an opportunity for further study on the relation between development communities and software architecture documentation. *(3)* An architecture is a direct reflection of its creator(s); by extension, the architecture document is a direct reflection of *its* creator(s). The personal knowledge, understandings, experiences, perspectives, interests, and biases become inextricably woven into the architecture documents. Again, this suggests that software architecture may be much more subjective at its heart than previously acknowledged.

Although the above factors directly influence how we think and go about designing and documenting software systems, they have not yet been given extensive treatment in software architecture and architecture description literature to date.

## 5.2 Quality and Credibility

As pointed out by Strauss and Corbin: "Some qualitative researchers maintain that the [standard validity criteria] by which quantitative studies are judged are quite inappropriate for judging the merit of qualitative studies" [46, p.266]. Instead, we prefer to talk of the accuracy and credibility of the research. Creswell proposes eight strategies for ensuring the accuracy of the results of qualitative studies [17, p.196]. Here we discuss how we implemented the six most relevant strategies.[5]

First, we used all three different types of *triangulation*: data, observer, and methodological [42, p.15]. In terms of data, we were able to check the statements the participants made in interviews against their questionnaire responses, the actual content of their chapters, and (for a subset of their statements) the input of the book editors. Both authors of this paper participated in the data collection and analysis, thus fulfilling the requirement for *observer* triangulation. Finally, some of our observations take into account quantitative aspects of the chapters (e.g., number of code fragments), which provides additional *methodological* triangulation.

We also used member-checking by asking participants for validation at two separate points in the process (validation questionnaire and final feedback). We used *rich descriptions* to the extent possible by phrasing our observations directly in the words of the participants. The *bias* that we bring is that both authors are software engineering professors who teach software architecture principles in a formal setting; this bias was transparent throughout the study. We also report all significant *discrepant information* that runs contrary to our themes. Finally, we asked one editor (Wilson) for feedback on a draft in lieu of a *peer debriefing* session.

[5]We omit *spend time in the field* and *use an external auditor*.

## 5.3 Limitations

The main limitations relate to the sampling, the specialized context, and the extent of the theme saturation. Our inevitable use of opportunistic sampling means that we may be unaware of some influential factors experienced by an author who did not participate in the study. Given the multi-case and exploratory nature of our study, the absence of such insights has limited potential to radically alter or contradict existing observations. However, it limits the breadth of what we can report. The specialized context (the writing of a book chapter) means that some of the observations may not apply directly to other contexts, such as reference documentation, or documentation produced in a corporate environment. Finally, as is usually the case with interviews, participants will be more passionate about certain aspects of their project than others, and may have little to say about some of the study themes. Our primary data collection method, the semi-structured interview, did not allow us to elicit an even amount of comments across, both, participants and themes. For this reason, some themes were developed based on the input of a limited number of participants. We mitigate this threat by focusing on the three primary themes and by systematically reporting the participants who supported each of our observations.

## 6. RELATED WORK

A number of software architecture books have emerged over the past two decades (e.g., [5, 11, 14, 21, 25, 26, 41, 44, 51]). Each of them has dealt with the issue of architectural documentation. On the whole, they advocate some combination of formal and semi-formal special-purpose notations (e.g., [44]), including UML (e.g., [26]), use of rigorously captured architectural patterns (e.g., [11]) and styles (e.g., [51]), representation of particular architectural views (e.g., [5]), and templates that mandate inclusion of specific information in a specific order (e.g., [25]). Overall, the advocated approaches were not followed by the AOSA authors.

The software architecture documentation concepts found in the above books are also discussed in many smaller publications that focus on architecture description languages (e.g., [13, 33, 37]), UML (e.g., [1, 32, 35]), architectural patterns and styles (e.g., [7, 30, 55]), architectural views (e.g., [9, 54, 31]), and standardized templates for capturing architectural knowledge (e.g., [19, 29, 50]).

The most comprehensive treatment of software architecture documentation is provided by Clements et al. [14]. They introduce "seven rules for sound documentation". These were partially followed by AOSA authors. For example, the AOSA chapters were written "from the reader's point of view", but did not necessarily "use a standard organization". Clements et al. describe four different architecture elaboration techniques: decomposition, uses, generalization, and layered. AOSA authors relied on these only informally and did not refer to them explicitly. The use of different architectural views and diagrams as suggested by Clements et al. is only sporadically followed by the AOSA authors. None of the authors used the modeling notations described by Clements et al. beyond the selective use of UML.

Several existing works focus specifically on the use of UML in practice. They help to inform some of the observations we made in our study. Fowler [24] suggests three different ways in which UML is used: as a design sketching aid, as a formal design notation, and as a programming language. Petre [40] conducted a study of the use of UML in practice where she found that UML was not widely used due to its complexity, lack of formal semantics, inter-view synchronization, and the resulting inconsistencies. This corroborated observations made in prior studies (e.g., [2, 3, 27, 28]), and reinforced the previously stated views [20, 32, 52] that UML's strengths (e.g., its wide applicability due to many views) are also a source of its observed weakness (e.g., its perceived complexity). This is also echoed in several of the responses we received in our study.

Several studies have also looked at social factors that influence architecture. For example, Waterman et al. [57] interviewed agile software developers to determine how much architectural design to conduct "up-front" as opposed to allowing it to emerge during development. The authors identified five different strategies—respond to change, address risk, emergent architecture, big design up-front, frameworks and template architectures—that can help an agile developer make this decision. AOSA authors touched upon several of these strategies in our study.

Tamburri and Di Nitto [48] have coined the term "social debt" to reflect the cost added to a project due to suboptimal architectural decisions that stem from issues connected to project personnel. The authors identify four sources of social debt: (1) separation of the system's architect from the developers; (2) different mindsets among architects; (3) architectural decisions that are implicitly captured across a sea of system artifacts; and (4) growing size of the development network. Each of these four sources was reflected in the decisions and responses of our study participants.

Su et al. [47] studied how researchers and practitioners "forage" documents when they are looking for architecturally-relevant information. They found that, on the whole, their participants looked for some subset of a system's purpose, actors, quality requirements, logical components, use cases, deployment of components, external dependencies, data persistence, and underlying platform. Su et al. also observed several sequences in which their subjects preferred to forage the architectural documents (e.g., quality requirements → components → use cases vs. components → use cases → deployment). To a large extent, this is reflective of the information and chapter organizations encountered in AOSA.

Our work is not the first to investigate knowledge capture and dissemination about a software system. Dagenais and Robillard [18] studied the role and impact of documentation in open-source projects. They found a link between the frequency of project documentation and code quality. They also observed the influence of the project community's involvement on the documentation, both positive (via regular interaction with the community) and negative (via the use of public wikis, which decrease the authoritativeness of the documentation). Our study refined and reinforced several of the findings as they pertain to software architecture.

Tang et al. [49] conducted an early survey to assess how architectural design rationale is documented by practitioners. This study uncovered that practitioners generally acknowledge the value of documenting architectural rationale, but do not always actually do it. Some of the reasons stem from the time constraints faced in software projects, and from the unclear cost of documenting rationale due to the lack of standardized approaches. Tang et al. found that practitioners tend to invent their own notations or introduce "home brewed" variations to existing notations. A similar observation was made by Baltes and Diehl [6], who found that the sketches and diagrams developers use to capture different aspects of a software system are typically informal, but occasionally include UML elements or use variants of UML. These observations were corroborated by our study.

## 7. CONCLUSION

With this study, we took an in-depth look at how 18 authors created essay-style descriptions (ESDs) of architectures for a set of open-source projects. The experiences we analyzed have three major implications.

First, as architectural ESDs are becoming a more common form of publicly accessible source of information on popular software systems (e.g., including the recent descriptions of ten such systems [53]), we can only assume that they will serve as a guide to an increasing number of software developers. By providing a rich account of the factors that influence the creation of ESDs, we hope to inform the future interpretation of these kinds of documents.

Second, by sharing a systematic interpretation of the lessons and experiences of highly-skilled open-source contributors, we hope to provide a basis for expanding our understanding of each ESD's context and underlying phenomena. In turn, this will serve as a foundation for planning future architectural ESD projects. For example, we discussed how the few AOSA authors who opted to use UML typically did so very informally, as a sketching tool. On the other hand, the authors of architectural ESDs in another recent project, DESOSA [53], relied on UML noticeably more and tended to include diagrams automatically generated by Eclipse from the systems' implementations. The structures of DESOSA chapters were also more uniform than is the case with AOSA chapters. Each of these differences can be attributed to the shared training the DESOSA authors received (in this case, as part of a university course).

Finally, many of the comments and rationales discussed in connection with the three novel themes we analyzed—*evolution*, *community*, and *personal factors*—have implications that go beyond the essay-style documentation format and may be worth exploring in the context of formal architectural documentation. On the one hand, it could be argued that traditional software architecture literature aims to facilitate and explain how to *document* a system's architecture for the benefit of system stakeholders, while the AOSA authors clearly also had to *communicate* their architectures broadly and effectively. On the other hand, insights such as the use of evolution stories to document rationale, or the impact of an author's experience on the selection of architectural concerns that are conveyed, would benefit from further study in the traditional setting. Ultimately, the boundary between essay-style architectural descriptions and formal documentation may not be as crisp as it can initially appear.

## 8. ACKNOWLEDGMENTS

This project was suggested by Greg Wilson. The authors are grateful to Greg Wilson and Amy Brown for their assistance and to the study participants for their enthusiastic contributions to software architecture and this study. Thanks also to Christoph Treude for valuable comments. This work was funded by NSERC and by NSF award 1117593.

## 9. REFERENCES

[1] M. Almorsy, J. Grundy, and A. S. Ibrahim. Automated Software Architecture Security Risk Analysis using Formalized Signatures. In *Proceedings of the 35th ACM/IEEE International Conference on Software Engineering*, pages 662–671, 2013.

[2] J. Aranda. *A Theory of Shared Understanding for Software Organizations*. PhD thesis, University of Toronto, Canada, 2010.

[3] J. Aranda, S. Easterbrook, and G. Wilson. Requirements in the wild: How small companies do it. In *Proceedings of the 15th IEEE International Requirements Engineering Conference*, pages 39–48, 2007.

[4] Association for Computing Machinery. The 2012 ACM computing classification system. http://www.acm.org/about/class/class/2012, 2012.

[5] M. A. Babar, T. Dingsoyr, P. Lago, and H. van Vliet (eds.). *Software Architecture Knowledge Management*. Springer, 2009.

[6] S. Baltes and S. Diehl. Sketches and diagrams in practice. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 530–541, 2014.

[7] J. M. Barnes, D. Garlan, and B. Schmerl. Evolution styles: foundations and models for software architecture evolution. *Software and Systems Modeling*, pages 649–678, 2014.

[8] A. Barth, C. Jackson, C. Reis, and The Google Chrome Team. The security architecture of the chromium browser. Technical report, Stanford University, 2008. http://seclab.stanford.edu/websec/chromium/.

[9] S. Brinkkemper and S. Pachidi. Functional architecture modeling for the software product industry. In *Proceedings of the 4th European Conference on Software Architecture*, pages 198–213, 2010.

[10] A. Brown and G. Wilson, editors. *The Architecture of Open-Source Applications*, volume 1 and 2. lulu.com, 2012. http://www.aosabook.org/en/index.html.

[11] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*. John Wiley & Sons, 2007.

[12] R. P. Buse and W. Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, 2010.

[13] C. Chapman, W. Emmerich, F. G. Marquez, S. Clayman, and A. Galis. Software architecture definition for on-demand cloud provisioning. *Cluster Computing*, 15(2):79–100, 2012.

[14] P. Clemens, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2nd edition, 2010.

[15] M. Conway. How do committees invent? *Datamation*, 14(4):28–31, 1968.

[16] J. Corbin and A. Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, 3rd edition, 2007.

[17] J. W. Creswell. *Qualitative Inquiry and Research Design*. Sage Publications, 2nd edition, 2007.

[18] B. Dagenais and M. P. Robillard. Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 127–136, 2010.

[19] K. A. de Graaf, A. Tang, P. Liang, and H. van Vliet. Ontology-based software architecture documentation. In *Proceedings of the 2012 IEEE Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture*, pages 121–130, 2012.

[20] D. Dori. Why significant UML change is unlikely. *Communications of the ACM*, 45(11):82–85, 2002.

[21] G. H. Fairbanks. *Just Enough Software Architecture: A Risk-Driven Approach*. Marshall & Brainerd, 2010.

[22] Firefox Team. Firefox OS architecture. https://developer.mozilla.org/en-US/Firefox_OS/Platform/Architecture, 2015.

[23] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[24] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, 3rd edition, 2003.

[25] I. Gorton. *Essential Software Architecture*. Springer, 2nd edition, 2011.

[26] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison-Wesley Professional, 1999.

[27] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristofferson. Empirical assessment of MDE in industry. In *Proceedings of the 33rd ACM/IEEE International Conference on Software Engineering*, pages 471–480, 2011.

[28] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristofferson. Model-driven engineering practices in industry. In *Proceedings of the 33rd ACM/IEEE International Conference on Software Engineering*, pages 633–642, 2011.

[29] ISO/IEC/IEEE. Iso/iec/ieee 42010:2011, systems and software engineering—architecture description. http://www.iso-architecture.org/42010/, 2011.

[30] A. Jansen, P. Avgeriou, and J. S. van der Ven. Enriching software architecture documentation. *Journal of Systems and Software*, 82(8):1232–1248, 2009.

[31] P. B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[32] C. F. Lange, M. R. Chaudron, and J. Muskens. UML Software Architecture and Design Description. *IEEE Software*, 23(2):40–46, 2006.

[33] N. G. Leveson, editor. *IEEE Transactions on Software Engineering – Special Issue on Software Architecture*. Volume 21, Issue 4, 1995.

[34] N. Medvidovic, E. Dashofy, and R. Taylor. Moving Architectural Description from Under the Technology Lamppost. *Journal of Information and Software Technology*, 49(1):12–31, 2007.

[35] N. Medvidovic, D. Rosenblum, D. Redmiles, and

J. Robbins. Modeling software architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology*, 11(1):2–57, 2002.

[36] N. Medvidovic, H. Tajalli, J. Garcia, Y. Brun, I. Krka, and G. Edwards. Engineering heterogeneous robotics systems: A software architecture-based approach. *IEEE Computer*, 44(5):62–71, 2011.

[37] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[38] R. Moreno-Vozmediano, R. Montero, and I. Llorente. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *IEEE Computer*, 45(12):65–72, 2012.

[39] D. E. Perry and A. L. Wolf. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[40] M. Petre. UML in practice. In *Proceedings of the 35th ACM/IEEE International Conference on Software Engineering*, pages 722–731, 2013.

[41] N. Rozanski and E. Woods. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional, 2nd edition, 2011.

[42] P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley and Sons, Inc., 2012.

[43] C. B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572, 1999.

[44] M. Shaw and D. Garlan. *Software Architecture: Perspectives on An Emerging Discipline*. Prentice Hall, 1996.

[45] D. Spinellis and G. Gousios, editors. *Beautiful Architecture: Leading Thinkers Reveal the Hidden Beauty in Software Design*. O'Reilly Media, 2009.

[46] A. Strauss and J. Corbin. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, 2nd edition, 1998.

[47] M. T. Su, E. Tempero, J. Hosking, and J. Grundy. A study of architectural information foraging in software architecture documents. In *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pages 141–150, 2012.

[48] D. A. Tamburri and E. Di Nitto. When software architecture leads to social debt. In *Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture*, pages 61–64, 2015.

[49] A. Tang, M. A. Babar, I. Gorton, and J. Han. A survey of the use and documentation of architecture design rationale. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, pages 89–98, 2005.

[50] A. Tang, P. Liang, and H. van Vliet. Software architecture documentation: The road ahead. In *Proceedings of the 2011 Working IEEE/IFIP Conference on Software Architecture*, pages 252–255, 2011.

[51] R. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2009.

[52] D. Thomas. MDA: Revenge of the modelers or UML utopia? *IEEE Software*, 21(3):22–24, 2004.

[53] A. van Deursen and R. Slag. DESOSA 2015: Delft students on software architecture. http://delftswa.github.io/, 2015.

[54] U. van Heesch, P. Avgeriou, and R. Hilliard. A documentation framework for architecture decisions. *Journal of Systems and Software*, 85(5):795–820, 2012.

[55] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The Koala component model for consumer electronics software. *IEEE Computer*, 33(3):78–85, 2000.

[56] B. Warsaw. *The Architecture of Open-Source Applications*, volume 2, chapter 10: GNU Mailman. lulu.com, 2012. http://www.aosabook.org/en/mailman.html.

[57] M. Waterman, J. Noble, and G. Allan. How much up-front? A grounded theory of agile architecture. In *Proceedings of the 37th ACM/IEEE International Conference on Software Engineering*, pages 347–357, 2015.

[58] R. K. Yin. *Case Study Research*. Sage, 2013.