Let's Go to the Whiteboard (Again): Perceptions From Software Architects on Whiteboard Architecture Meetings

Eduardo Santana de Almeida, *Senior Member, IEEE*, Iftekhar Ahmed, and André van der Hoek, *Member, IEEE*

***Abstract*—The whiteboard plays a crucial role in the day-to-day lives of software architects, as they frequently will organize meetings at the whiteboard to discuss a new architecture, some proposed changes to an existing architecture, a mismatch between a prescribed architecture and its code, and more. While much has been studied about software architects, the architectures they produce, and how they produce them, a detailed understanding of these whiteboards meetings is still lacking. In this paper, we contribute a mixed-methods study involving semi-structured interviews and a subsequent survey to understand the perceptions of software architects on whiteboard architecture meetings. We focus on four aspects: (1) why do they hold these meetings, (2) what is the impact of the experience levels of the participants in these meetings, (3) how do the architects document the meetings, and (4) what kinds of changes are made in downstream activities to the work produced after the meetings have concluded? In studying these aspects, we identify eleven observations related to both technical aspects and social aspects of the meetings. These insights have implications for further research, offer concrete advice to practitioners, and suggest ways of educating future software architects.**

***Index Terms*—Software architecture, software architects, whiteboard meetings, architecture documentation, interviews, survey.**

I. Introduction

DESIGNING a software architecture is not purely a technical issue [1], [2]. It also involves numerous social, human, and organizational aspects [3], [4] that can influence the success of an architecture, and thus the entire project, considerably. One such aspect concerns who is involved in the design process: the different stakeholders participating must be selected with care [5], [6]. So it is with the primary software architect or architects leading the effort: they are typically experienced members of the team who are ultimately responsible for the design choices made, validating them, and capturing and sharing them in various kinds of artifacts that will be used downstream [7].

To date, the software engineering community has studied which kinds of projects and organizations need a software architect [1], the assigned duties, skills, and knowledge of architects [2], what software architects actually do [7], [8], their mindset [9], the reasoning process involved in making architectural decisions [10], [11], and the impact of architects writing code themselves [12], [13], among others. Not all findings from these studies are uniform. For instance, what architects do varies considerably from one organization to another and even from one project to another within the same organization [13], [14]. Moreover, gaps exist in our collective understanding, which has led to calls for further studies of software architects, their work, and how they conduct it [15], [16], [17].

This paper contributes one such study, with two closely related goals. Our first goal is to understand how software architects perceive the whiteboard design meetings in which they participate. Whiteboards are an important medium used in reasoning and problem solving in many disciplines, such as civil engineering [18], mechanical engineering [19], and design engineering [19]. In software engineering, previous whiteboard studies focused on programmers [6], [20], [21] and software designers [22], [23], but not architects. With software architecture design laying the critical foundations for a project's success, then, it is important to study software architecture design meetings at the whiteboard. While not all architecture design work takes place at the whiteboard, it is anecdotally well-known—and our study confirms—that architects use the whiteboard for a broad range of reasons. Understanding the unique context, demands, and approaches of this setting is a primary goal of this paper.

Our second goal is to understand the transition from architectural design sketches as produced in whiteboard software architecture meetings to downstream development activities. Architecture work at the whiteboard is necessarily incomplete and, when the rubber hits the road during additional architectural refinement, more detailed design, or implementation, changes to the architecture as first envisioned are frequently needed [23]. While much has been said about the need for architecture and implementation to stay in sync [24], as

Manuscript received 27 October 2022; revised 14 August 2023; accepted 6 September 2023. Date of current version 17 October 2023. This work was supported in part by the National Science Foundation under Grant CCF-2210812; INES (www.ines.org.br); CNPq under Grant 465614/2014-0; CAPES under Grant 88887.136410/2017-00; FACEPE under Grants APQ-0399-1.03/17 and PRONEX APQ/0388-1.03/14; and FAPESB under Grant INCITE PIE0002/2022. Recommended for acceptance by Y. Cai. *(Corresponding author: Eduardo Santana de Almeida.)*

Eduardo Santana de Almeida is with the Institute of Computing (IC-UFBA), Federal University of Bahia, Salvador, Bahia 40.170-110, Brazil (e-mail: esa@rise.com.br).

Iftekhar Ahmed and André van der Hoek are with the Irvine Donald Bren School of Information and Computer Sciences Department of Informatics, University of California, Irvine (UCI), Irvine, CA 92697 USA.

Digital Object Identifier 10.1109/TSE.2023.3314410

well as about architectural erosion and drift over time [25], no study to date has examined how information from whiteboard architecture design meetings makes its way downstream, what kinds of deviations happen in this process, and why.

We conducted semi-structured interviews with 27 software architects from 18 different companies across five different countries, with the software being worked on by these architects spanning a wide range of application domains. We then surveyed an additional 46 software architects across nine countries to contextualize and augment the findings from the interviews. Our focus in conducting these interviews and surveys was on the aforementioned two goals. We could have also included an analysis of the architectural sketches and how they are created on the whiteboard or an examination of how the participants engage in design discussions surrounding the sketches, but, as exemplified by [26], each should be a separate study in its own right to gain meaningful depth. We thus center on the following four research questions:

- **RQ1:** Why do software architects hold whiteboard software architecture design meetings? Design work is varied and may take place across a range of meetings, some at the whiteboard, some elsewhere [27]. As a base for the subsequent research questions, we seek to document what architectural design activities take place in whiteboard meetings.
- **RQ2:** What is the impact of the experience of the participants on these meetings? Because of the complexities involved and stakes of architectural design, it is often thought to be the domain of experts [7]. Whiteboard design meetings, however, do not only involve experts. We seek to understand how the mix of participants impacts how design work in proceeds in these meetings.
- **RQ3:** How do software architects document what happens in whiteboard software architecture meetings? Anecdotally, a variety of mechanisms are used, including relying on memory, taking photographs, or assigning notetakers [21]. We seek to build a more comprehensive understanding of the variety of approaches used.
- **RQ4:** What kinds of changes arise in downstream activities to the work produced in architecture whiteboard meetings? Any designed architecture is subject to further updates and changes after it has been produced and documented. We seek to identify the reasons why architects believe such updates and changes are made.

Together, the answers to these questions have implications for on the one hand research and what to study next and on the other hand practice in how to best approach these meetings and factors an organization or team might consider in improving its own architecture whiteboard meetings.

This paper constitutes the first broad empirical study of how software architects view and engage in whiteboard architecture meetings, as well as the outflow of those meetings into downstream activities. It makes the following contributions:

- A mixed qualitative and quantitative study that investigates key aspects of whiteboard software architecture meetings, with a primary focus on the role of experience and how the results from these meetings transition to later activities.
- A set of eleven observations regarding the nature of whiteboard software architecture meetings, covering concrete technical, human, and social perspectives on these meetings and offering recommendations for practitioners, researchers, tool builders, and educators.
- A collection of all our research materials on a project website for replication and reproducible research purposes, including our interview data (prompts, transcriptions, and codebook) and the survey instrument.

II. Related Work

A. Studies of Sketching and Whiteboard Use

Dekel and Herbsleb conducted an observational study analyzing several software design meetings from the ACM DesignFest event that was held at the 2005 OOPSLA conference [28]. The analysis of the meetings primarily focused on the notations that the designers used and the representations they created in those notations, detailing for instance how the designers often started with unstructured representations and how they at times combined content from what were independently created diagrams into a single diagram. Based on the results of the study, Dekel and Herbsleb discussed several implications for the design of future tools supporting whiteboard based design. Cherubini et al. [21] performed an exploratory study at Microsoft of how and why developers use whiteboards, with a particular focus on how developers 'draw their code'. Based on interviews and surveys with developers, they found that informal notations were used in support of face-to-face communication about the code and that available modeling tools were not capable of supporting this need since their focus on formal, correct diagrams does not match the informal nature in which developers seek to externalize their mental models of code.

Leveraging one of the videos collected for the 2009 Studying Professional Software Design workshop [26], Nakakoji et al. examined the conversations and whiteboard drawings of a pair of professionals with the help of the Design Practice Streams (DPS) tool they developed [29]. DPS allows replays of strokes on the whiteboard and connects the replay to an automatically created transcript of the meeting, enabling quick exploration of concepts and when they were talked about. In the case of the video analyzed, Nakakoji et al. highlight the role of key concepts and what aspects of the design being worked on were most frequently re-discussed.

In order to understand how to provide improved tool support for integrating visual sketches into developers' workflows, Walny et al. conducted a qualitative study centered on the creation, use, and transformation of sketches [30]. Using semi-structured interviews with eight software developers, their particular focus was on the lifetime of sketches: with what medium they were created first (e.g., paper, whiteboard, tool) and how they then were captured, augmented, and re-created in similar and other media (e.g., photo, tablet, another piece of paper).

Baltes and Diehl investigated the use of sketches and diagrams in software engineering practice, with a particular focus on their relation to source code artifacts [31]. Using data from three companies and a survey, they found that the majority of

the sketches were informal and that the most common purposes for creating sketches and diagrams were designing, explaining, and understanding. More than half of the sketches were created on analog media like paper or whiteboards and were revised after creation. Based on the findings from this work, Baltes et al. developed SketchLink, a tool that aims at increasing the value of sketches and diagrams by explicitly linking them to the source code to which they pertain [32].

Mangano et al. conducted an observational study analyzing fourteen hours of design activity by eight pairs of professional software developers at the whiteboard [23]. In the study, each pair was provided with a written prompt asking them to design an educational traffic light simulation program. The researchers analyzed the type of sketches created, how professional software designers focus on individual sketches and shift their attention among sketches, and the reasoning process to understand and advance the state of the design at hand.

Out of these prior studies, the ideas discussed in [21], [22], and [31] are most closely related to our study. However, our study is unique in focusing on software architects and their design activities at the whiteboard, as well as in examining important aspects not covered in previous work, such as the influence of levels of experience and the changes architectures created at the whiteboard undergo in downstream activities.

Beyond studies that specifically focus on creating an understanding of sketching and whiteboard use, many tools have been proposed to explicitly support software developers in their design sketching. Baltes et al. [33], for instance, presented LivelySketches, a prototype tool that supports the round-trip lifecycle of sketches from analog to digital and back. As another example, FlexiSketch supports collaboration across distances, with the ability for users to define notations on the fly [34]. An interesting recent example seeks to integrate sketching in the IDE [35]. Compared to these and many other tools that have been proposed, our paper does not contribute any tool designs, though our findings give rise to implications for future tools.

B. Studies of Software Architects

Personal Experiences. The Pragmatic Architect column in the IEEE Software magazine discussed many aspects about the software architect and their role over the years. In [1], for instance, Fowler introduced key definitions of software architecture and the architect's role in creating and managing them. Buschmann [36] shared and commented on a compressed week-long diary of what his 'real life' as an architect is like when working on a product line for an industrial automation system, which is similar to Kruchten [7] who described what software architects 'really do' based on his experience in managing a 10-person architecture team from 1992–1995. Drawing on more than ten years working as a software architect, Woods [37] classified architects into three groups: enterprise, infrastructure, and application architects. Klein [38] discussed what makes a software architect successful. Erder and Pureur [17] considered the architect's role in agile development and followed up on this discussion in a later paper that examined desired personality traits of software architects [39]. Woods [12] discussed the benefits and drawbacks of architects actually engaging in programming beyond their primary role as a designer. Klein [40], based on his experience in industry, defined a three-phase model (Blank Page, Integration, and Magic) to capture the evolution of software systems, and discussed the kinds of contributions necessary from the software architect for achieving success in each phase. Sarang [41] proposed a structure for an architecture team and defined the roles and responsibilities of the members.

Surveys. Clements et al. [2] investigated the human aspects of architecting software, focusing on the duties, skills, and knowledge of software architects. They canvased over 200 public sources of information (e.g., web sites, blogs, training and education materials, job descriptions) to identify about 200 different duties, 100 skills, and 100 areas of knowledge – each of which was mentioned by at least one source.

Clerc et al. [9] performed a survey in the Netherlands to collect feedback on the importance of architectural knowledge for the daily work of practitioners in architecture. Based on the answers from 107 respondents, the study provides insights in the way practitioners view and use architectural knowledge by listing which uses are important for the different roles that architects play (e.g., project lead, reviewer, consultant) and on what architectural level (e.g., software, information, enterprise).

Heesch and Avgeriou [10] surveyed 53 software architects from several companies and project domains to get insights in the reasoning processes followed in architectural design. Among a variety of findings, they show that architects typically are involved in requirements elicitation and therefore understand the reasoning behind the requirements well, that architects find it important to search for multiple options but equally consider this an expensive activity and only engage in doing so when truly necessary (thus favoring known solutions), and that architects seldom reject decisions they have made before.

Hoorn et al. [8] conducted a large-scale survey with 142 software architects from four IT organizations in the Netherlands to understand what architects do on a day-to-day basis and what kind of support they need for sharing architectural knowledge.

Case Studies. Premaj et al. [42] conducted a case study with two projects by performing retrospective root cause analyses into the issues assigned to software architects to understand why the issues arise, what types they are, and how their occurrences could potentially be reduced in future through improvements in the development process.

Rehman et al. [13] conducted a two-stage case study, combining data analytics for five open source projects with semi-structured interviews of several architects of these systems. The authors addressed three questions: Do architects write code? What type of code do architects write? Is there any empirical evidence to support that software projects will benefit from hands-on software architects? The primary conclusion is that benefits exist to architects writing code.

While the focus of our study is different from these existing studies of software architects, the viewpoints expressed in the personal experiences and the results from the case studies were particularly influential in shaping the direction of our study to provide a complementary view of an important under-understood activity: whiteboard architectural design together

with its downstream outflow. In addition, the surveys conducted in [2], [8], [9], [10], [42], and [8] served as inspiration for how we structured our survey questions.

III. Research Design

We adopted a two-part research design for our study: we first conducted in-depth semi-structured interviews with experienced software architects and then performed a validation survey with additional, again experienced, software architects. The goal of the interviews was to gather insights into practitioner views to help us to formulate a clear picture of whiteboard software architecture meetings and the outflow from these meetings. These insights were then checked and refined by the results from the validation survey. All study materials from the interviews and surveys can be found in the supplementary materials for the paper.[1]

A. Interviews

Protocol. We interviewed 27 software architects with experience in whiteboard software architecture meetings. The first author interviewed the software architects either in person, if they worked in the same area, or via Skype, if they did not. The average length of the interviews was 37 minutes, the median 16.49, the shortest 7.18, and the longest 43.42.

Each interview consisted of two parts. In the first part, in addition to a few demographic questions and questions about the experience level of the interviewee, the interviewer asked open-ended questions about their engagement in whiteboard architecture design meetings, their perceptions about the participation of novice and experienced software architects, and their advice for other software architects participating in these kinds of meetings. In the second part, the interviewer asked questions related to the importance of the meetings to implementation, approaches used to document and communicate outcomes, decisions and structures preserved from whiteboard to code, typical changes that take place when moving from sketched designs to concrete implementation, and aspects missing from whiteboard discussions. Finally, we thanked the interviewees and debriefed them by informing them about what we planned to do with the data. The protocol was designed by two researchers over a period of three months. Throughout the interviews, we limited the conversation to in-person whiteboard meetings.

Participants. We selected the software architects to be interviewed based on convenience sampling and snowballing. We directly invited twenty software architects with experience in designing software architectures and participating in whiteboard architecture meetings. All of them agreed to be interviewed. Each of them was asked for recommendations for other potential interviewees, which led to additional participants whom we asked the same question. This eventually resulted in twenty-seven software architects who agreed to participate in the study.

We first conducted a pilot interview with another software architect (not included in the study) as a pretest [43] and then performed the interviews with the 27 architects. The software architects came from eighteen different companies across five different countries (Brazil 13, Canada 1, Germany 4, Sweden 2, and USA 7), working across a range of different domains, including but not limited to games, music, finance, e-commerce, data science, and sports. Note that after 24 interviews, we reached saturation. We performed a few extra interviews to ensure this was the case, reaching a total of 27. All interviews were conducted by the first author over a period of four months. Only one interview was completed face to face.

TABLE I
Summary of Professional and Demographic Information of Participants

SA	Experience	Domain	Country
SA 1	2–5 years	IT	Brazil
SA 2	5+ years	e-Government	Brazil
SA 3	2–5 years	e-Commerce	Brazil
SA 4	5+ years	IT	Brazil
SA 5	5+ years	IT	Brazil
SA 6	2–5 years	Marketing	Brazil
SA 7	5+ years	Audio streaming	Sweden
SA 8	5+ years	IT	Brazil
SA 9	5+ years	IT	Brazil
SA 10	5+ years	Finance	Brazil
SA 11	5+ years	Marketing	Brazil
SA 12	2–5 years	Embedded Software	Germany
SA 13	5+ years	Finance	Brazil
SA 14	5+ years	e-Commerce	Canada
SA 15	5+ years	Embedded Software	USA
SA 16	5+ years	e-Government	Brazil
SA 17	5+ years	Embedded Software	USA
SA 18	5+ years	e-Government	Brazil
SA 19	5+ years	Games	Sweden
SA 20	5+ years	Cyber-physical	USA
SA 21	5+ years	Embedded Software	Germany
SA 22	5+ years	Embedded Software	USA
SA 23	5+ years	Embedded Software	Germany
SA 24	5+ years	Embedded Software	USA
SA 25	5+ years	Embedded Software	Germany
SA 26	5+ years	Embedded Software	USA
SA 27	5+ years	Sports	USA

In the remainder of the paper, we label each interviewee SA1 through SA27. Table I provides a summary of the architects' backgrounds in terms of their years of experience working as a software architect, business domain in which they develop software, and country where they reside and work. The supplementary material attached to the paper presents detailed (though anonymized) information about the participants.

Data analysis. Data analysis started with audio transcription (16 hours and 46 minutes total). Two researchers conducted the transcription using the Trint tool[2] and, after internal review, all interview transcripts were formatted and then shared with the respective software architects for validation; this resulted in minor adjustments involving the architects providing clarifications of some of their answers.

After that, the coding process was started by two researchers using the NVivo tool[3]. The two researchers had previous experience in conducting studies combining interviews, albeit in different aspects of software engineering. Because the interviews were open-ended conversations with a limited number of predefined lead questions and a broad range of open-ended

[1] https://github.com/whiteboard-architecture/empirical-study

[2] https://trint.com

[3] https://www.qsrinternational.com/nvivo/nvivo-products

and spontaneous follow-up questions by the researchers, and because interviewees at times would return to prior topics as part of answering later questions, we used a set of first cycle and second cycle coding methods to data analysis [44]. First cycle methods are those processes that happen during the initial coding of data, where, per question, we identified and preliminarily tagged possible relevant excerpts in the transcripts using open coding [44]. After the first few transcripts, the researcher met and created a unified coding schema, which they then applied independently to the remainder of the transcripts. Once each researcher had performed their independent turn on the remainder, the two authors met to resolve differences in coding through negotiated agreement [45].

Second cycle methods are advanced ways of reorganizing and reanalyzing data coded by first cycle methods; in our study, the codes created in the first cycle were refined and clustered by the two researchers together into distinct, conceptually coherent categories with associated excerpts. These categories, then, served as the source of our results as discussed in Section IV.

Note that we did not strive to develop a theory at this time. Doing so is not always a necessary outcome for a qualitative study [46] and, in our case, is actually ill suited for the research questions we pose. Rather, we seek to document the varied perceptions and experiences of software architects working in the particular context of whiteboard meetings.

B. Survey

Protocol. Based on the results from the interviews, we designed a 30-minute survey to further build our understanding of the perceptions of software architects on whiteboard software architecture meetings. The first draft of the survey was composed of seventeen questions, fifteen of which were closed questions and two of which were open questions. For the design of the survey, we followed Kitchenham and Pfleeger's guidelines for personal opinion surveys [47]. As one of the guidelines, previous surveys related to sketches in software engineering [31] and software architects [2], [8], [9], [10] were consulted.

We piloted our survey with two experienced software architects to get feedback on the formulation of the questions, difficulties faced in answering the survey overall, and time to finish it. As these pilot respondents were experts in the area, we also wanted to know whether they felt we were asking the right kinds of questions or should be changing the approach. In response to their feedback, we modified the survey several times, rephrasing some questions and removing others to make it easier to understand and answer. The final version of the survey consisted of 23 questions (including demographics). The pilot survey responses were used solely to improve the questions, and these responses were not included in the final results. We kept the survey anonymous, but in the end, the respondents could share their email to receive a summary of the study. The survey instrument is included in the supplementary material.

Participants. We followed a three-step approach to recruit survey respondents: initially, we posted survey information on personal accounts on social media (e.g., Twitter, LinkedIn). Next, two authors contacted potential respondents by email (convenience sample) and asked them to share it with other potential respondents (snowballing). Because of this process, we were not able to track the total number of invitations. Overall, we received 50 responses, out of which we disqualified four responses that did not have any responses to any of the actual survey questions of interest (despite having responses to the basic demographics questions). This led to 46 valid responses that were considered where the survey respondents answered all questions.

The respondents were spread out across nine countries and four continents. The top three countries where the respondents came from were United States, Brazil, and Germany. The professional experience of the 46 respondents working as software architects varied from one year to 31 years, with an average of 14 years and a median of 15 years. The majority of the respondents had an advanced degree (67.4%), i.e., Master's or Ph.D., 30.4% of the respondents had a Bachelor's degree, and 2.2% graduated from high school without completing college.

Data Analysis. We collected the ratings that our respondents provided for each closed question and converted these ratings to Likert scores from 1 (Strongly Disagree) to 5 (Strongly Agree). We computed the average Likert score of each statement related to different perspectives (e.g., reasons to conduct whiteboard architecture meetings, different levels of experience, role of documentation, downstream changes, and digital tools) and plotted Likert scale graphs. In addition, we used open coding to analyze the answers that the survey respondents gave to the two open questions related to recommendations for software architects conducting whiteboard architecture meetings and final thoughts on the topic. To reduce subjective bias during the open coding process, we assigned both to two authors of this paper. Each author analyzed the answers separately. Once all the data were coded, the two authors met to resolve differences in coding.

IV. Results

In this section, we present the results for each of the four research questions identified in Section I. Before we do so, however, we first further characterize whiteboard software architecture meetings based on the responses from the architects.

A. Whiteboard Software Architecture Meetings

During the interview sessions, we asked the participants to give an example or two of recent whiteboard software architecture meetings in which they participated. We specifically asked them to describe the setting, how many participants there were, what roles they held, what problem the group addressed, and how long the meeting lasted.

The meetings involved from two (min) to 15 (max) participants, with five being the average. Beyond the interviewed software architects themselves, other participants held many different roles, including other software architects, developers, requirements engineers, test engineers, marketing managers, product managers, consultants, UX engineers, and users. According to one of the software architects (SA 10): *"... you have a team and everyone is working together, so the developer who is doing the code is participating in the architecture*

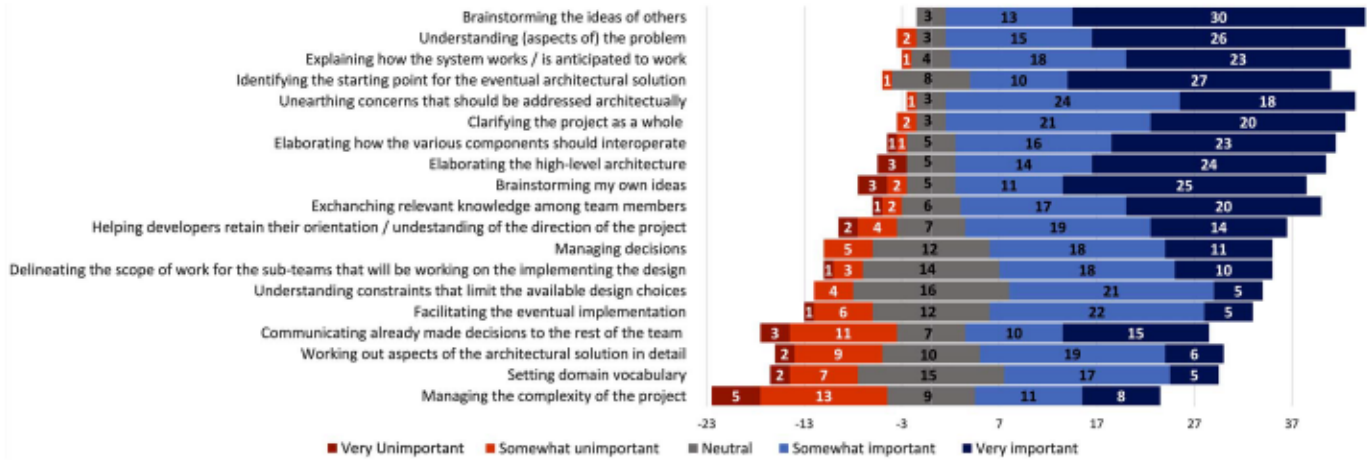

Fig. 1. Nineteen reasons for software architects to conduct whiteboard software architecture meetings.

discussion. The tester is also participating, the requirements analyst too, everyone is on the same boat. There is no longer separation."

The topics discussed ranged from system integration and service design, through cloud deployment and performance, to knowledge management and even pre-sales. As one example, a software architect (SA 21) described the topic of a whiteboard meeting concerning the performance in the face of scale of the software for which they are responsible as follows: *"We're using Lambda functions to go from one step to another and we are seeing performance challenges on processing large files given the limitation of the Lambda function of having five hundred MB per data frame, so you can open any file that will consume less than half GB memory of RAM and then do the processing."* As another example, another architect (SA 7) described a meeting they organized as: *"As we had a system that exchanged many messages over the network and needed very high performance, several times we discussed architectural problems that could bring I/O bottlenecks, because it was a system that wrote a lot on disk and sent many messages over the network, so we had I/O bottlenecks on the machine running, as network bottlenecks needed high-level architectural discussions."*

The meetings ranged from 20 minutes (minimum) to seven hours (maximum), with an average of 1 hour and 10 minutes. According to one of the software architects (SA 7), meetings take longer when they involve activities that expressly seek to document outcomes: *"These meetings are roughly I would say between four and seven hours. And this other meeting that I told you about where we are documenting. So, when do we do that, we typically also reserve at least half a day or even better a day. So that we spend also at least four to six hours really working on it."* This clearly is toward the extreme end of meeting duration. At the same time, it recognizes that architecture work is not easy and requires participants to engage in depth to work through what sometimes can be very complex issues. In this case, the culture at the company at which the architect works is such that longer meetings to sort things out are preferred over spreading the work across multiple, more disconnected meetings.

B. Reasons to Conduct Whiteboard Software Architecture Meetings (RQ1)

From the interviews, we identified 19 different reasons why software architects go to the whiteboard and hold meetings concerning software architecture. We used the survey responses to rank these 19 reasons, with Fig. 1 showing the results. The top five reasons are: *brainstorming the ideas of others* (average Likert score for this statement is 4.59, i.e., between "somewhat important" and "very important"), *understanding (aspects of) the problem that the architecture has to solve* (average Likert score for this statement is 4.54, i.e., between "somewhat important" and "very important"), *explaining how the system works/is anticipated to work* (average Likert score for this statement is 4.37, i.e., between "somewhat important" and "very important"), *identifying the starting point for the eventual architectural solution* (average Likert score for this statement is 4.37, i.e., between "somewhat important" and "very important"), and *unearthing concerns that should be addressed architecturally* (average Likert score for this statement is 4.28, i.e., between "somewhat important" and "very important").

The following are some comments from the software architects (SA) that highlight these aspects, together with what reason it was classified as:

✓*SA 3: "Most of the time that an important decision was to be made or some doubt existed related to some important decision, we used the whiteboard to discuss or brainstorm things or draw some things that were the common understanding of the team."* [understanding (aspects of) the problem that the architecture has to solve]

✓*SA 13: "So I would say that the solution crystallizes on the whiteboard. The whiteboard makes it possible to quickly change the solution or to brainstorm some different aspects. So, the brainstorming is I would say impossible without the whiteboard".* [brainstorming the ideas of others]

In our survey, we also asked respondents about the importance of whiteboard architecture meetings to successful architectural design. Among the 46 survey respondents, 56.52 percent (26 respondents) consider whiteboard design meetings very important to successful architectural design, 41.30 percent

TABLE II
IMPORTANT ASPECTS OF EXPERIENCE RELEVANT TO WHITEBOARD SOFTWARE ARCHITECTURE MEETINGS

Meaning of Experience	Frequency
Design of previous architectures	5
Ability to communicate	4
Participation in previous projects	3
Technical knowledge	3
Domain knowledge	3
Architectural knowledge on methods and tools	2
Awareness of technology trends	1
Ability to think strategically	1
Blend of technical and non-technical aspects	1
Depth of knowledge about a theme with breadth of knowledge overall	1
Facilitation of meeting discussion	1
Having faced failures	1
Good understanding about existing systems	1
Ability to interact with project manager	1
Many hours of development	1
Knowing a lot of abstractions across domains	1

(19) ranked the meetings as important, and 2.17 percent (1) were neutral.

Observation 1

Brainstorming the ideas of others, understanding (aspects of) the problem that the architecture has to solve, explaining how the system works/is anticipated to work, identifying the starting point for the eventual architectural solution, and unearthing concerns that should be addressed architecturally are the top five reasons for software architects to conduct whiteboard software architecture meetings.

C. Experience (RQ2)

1) Important Aspects of Experience: According to Kruchten [7], a software architect is a software expert responsible for designing, developing, nurturing, and maintaining the architecture of the software-intensive systems in which they are involved. Kruchten further observes that, in general, the architect's role is typically reserved for someone with significant experience in prior projects.

In order to better understand what kind of experience is relevant to whiteboard software architecture meetings, we asked the software architects during our interviews. Collectively, they identified sixteen different aspects, as listed in Table II with the number of architects that identified each aspect. The top five aspects are: *design of previous architectures* (5), *ability to communicate* (4), *participation in previous projects* (3), *technical knowledge* (3), and *domain knowledge* (3).

The following are some comments from some of the software architects that highlight these aspects, together with the corresponding meaning of experience:

✓*SA 9: "When we are with experienced architects, they have already designed and implemented various software and have more notion of what works and does not work, they have already seen several examples of architecture, so that is what they bring of importance, the experience itself."* [design of previous architectures]

✓*SA 28: "So I think there is probably two aspects to the experience, right. One is: experience with how to communicate, right. So how to, how to produce design on the fly and communicate that to the other people in the room quickly."* [ability to communicate]

✓*SA 7: "Experience is also defined in terms of the variety of projects. If a person has developed an embedded system, a mobile system, a web system, a large-scale micro-controller system, in various contexts, [ed: they] will be able to compare very well the performance of a web system is different from mobile performance, embedded, etc."* [participation in previous projects]

Overall, while the aspects mentioned differ, we note that the majority point to the need for a strong technical background that is not necessarily limited to just one project or type of project.

Observation 2

Design of previous architectures, ability to communicate, participation in previous projects, technical knowledge, and domain knowledge are the top five aspects of experience seen as useful to whiteboard software architecture meetings.

2) Team Composition: Most companies have a limited number of software architects and depending on the size of the company, it may have just one or two people in this role. Thus, whiteboard software architecture meetings necessarily not only involve participants in different roles (as mentioned in Section IV-A), but also participants who are likely to have different levels of experience. In the interview sessions, we asked about these different levels of expertise as they relate to team composition for meetings at the whiteboard, and we used the surveys to gather additional information in this regard.

We first discuss the participation of novices. The interviewees mentioned that novices typically were invited for two primary reasons. First, even though they were novices, they sometimes already were responsible for some (small) part of the code base that might be influenced by the architectural changes being discussed. As such, the novices needed to be there and participate just as more experienced developers. The second reason is learning: by inviting novices to the meetings, the architects provided the novices with the opportunity to become familiar with other parts of the code as well as the higher level architectural structures within which their parts of the code fit. In this case, participants were still encouraged to participate and ask questions, though they otherwise did not carry critical responsibility. Rarely did the novices lead the meeting, which is understandable because it is the architects that we interviewed who hold full responsibility for the architecture and the decisions governing its evolution over time.

The interviewees shared eleven different perceptions on including novices in the meetings, which were ranked by the software architects who participated in the survey (Fig. 2). The top five perceptions are: *with novices, the team has to provide more context and offer more explanation during whiteboard software design meetings* (average Likert score for this statement is 4.28, i.e., between "somewhat agree" and "strongly

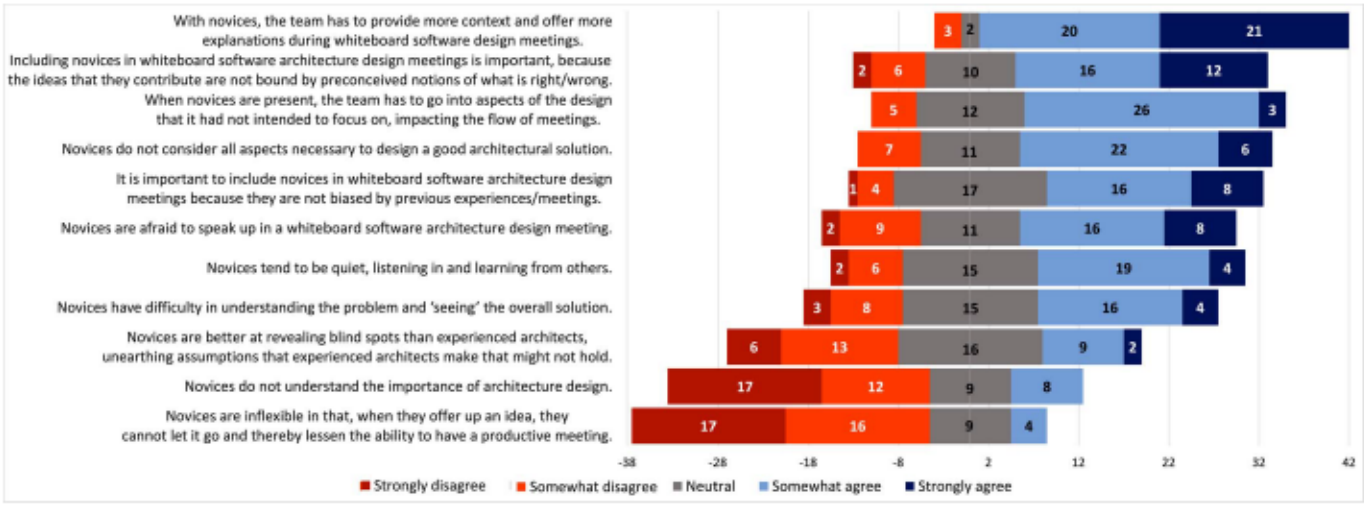

Fig. 2. Architects' perceptions on including novice participants in whiteboard software architecture meetings.

agree"), *including novices in whiteboard software architecture design meetings is important, because the ideas that they contribute are not bound by preconceived notions of what is right/wrong* (3.65, between "neutral" and "somewhat agree"), *when novices are present, the team has to go into aspects of the design that it had not intended to focus on, impacting the flow of meetings* (3.59, between "neutral" and "somewhat agree"), *novices do not consider all aspects necessary to design a good architectural solution* (3.59, between "neutral" and "somewhat agree"), and *it is important to include novices in whiteboard software architecture design meetings because they are not biased by previous experiences/meetings* (3.57, between "neutral" and "somewhat agree"). The following are some comments from the interviewees that highlight these aspects:

✗*SA 18: "When it is with more inexperienced architects, sometimes it tends to be a bit more for lecture. Some points you end up having to go deeper to see if you bring the person to the same level or tend to go down. The person still does not make a clear division between what is architecture level and implementation. It goes down and up much more often and you have to keep pulling the person up again."* [with novices, the team has to provide more context and offer more explanations during whiteboard software design meetings]

✓*SA 23: "So having novices in the room who are unafraid to ask questions can help to clarify things which often leads to actual insights that would have gotten glossed over had they not been written down."* [novices are better at revealing blind spots than experienced architects, unearthing assumptions that experienced architects make that may not hold]

✗*SA 12: "The beginner has a lot of difficulty, sometimes, to see the whole solution. The beginner is very focused on the use of technology and a little distant from the solution as a whole. This is my perception. They already want to discuss the technology, the infrastructure, the implementation, they have a very great anxiety, the more novice the developer. The tendency is to try to contain these impetus and take off the source code leading to discussion at the architecture level, regardless of technology or how it will be used. That's my main difficulty with new developers. They are still very much attached to the code and technology that will be used in the project."* [novices have difficulty in understanding the problem and 'seeing' the overall solution]

Observation 3

The top five perceptions on including novices in whiteboard software architecture meetings are: with novices, the team has to provide more context and offer more explanations during whiteboard software design meetings; including novices in whiteboard software architecture design meetings is important, because the ideas that they contribute are not bound by preconceived notions of what is right/wrong; when novices are present, the team has to go into aspects of the design that it had not intended to focus on, impacting the flow of meetings; novices do not consider all aspects necessary to design a good architectural solution; and It is important to include novices in whiteboard software architecture design meetings because they are not biased by previous experiences/meetings.

Note that the various perceptions represent a mix of positive and negative effects of including novices, so the order in which the perceptions are placed in Fig. 2 should not be read as ranging from positive perceptions (top) to negative perceptions (bottom), or vice versa. Rather, the figure is ordered by level of agreeableness. The observation that, with novices, the team has to provide more context and offer more explanation during whiteboard software design meetings was agreed to most often, while the observation that novices are inflexible in that, when they offer up an idea, they cannot let it go and thereby lessen the ability to have a productive meeting, was agreed to least often.

In our survey, we also asked respondents whether they felt that overall it is important to include novices in whiteboard software architecture design meetings. Out of 46 respondents, 23 respondents strongly agree, 19 somewhat agree, and 2 are

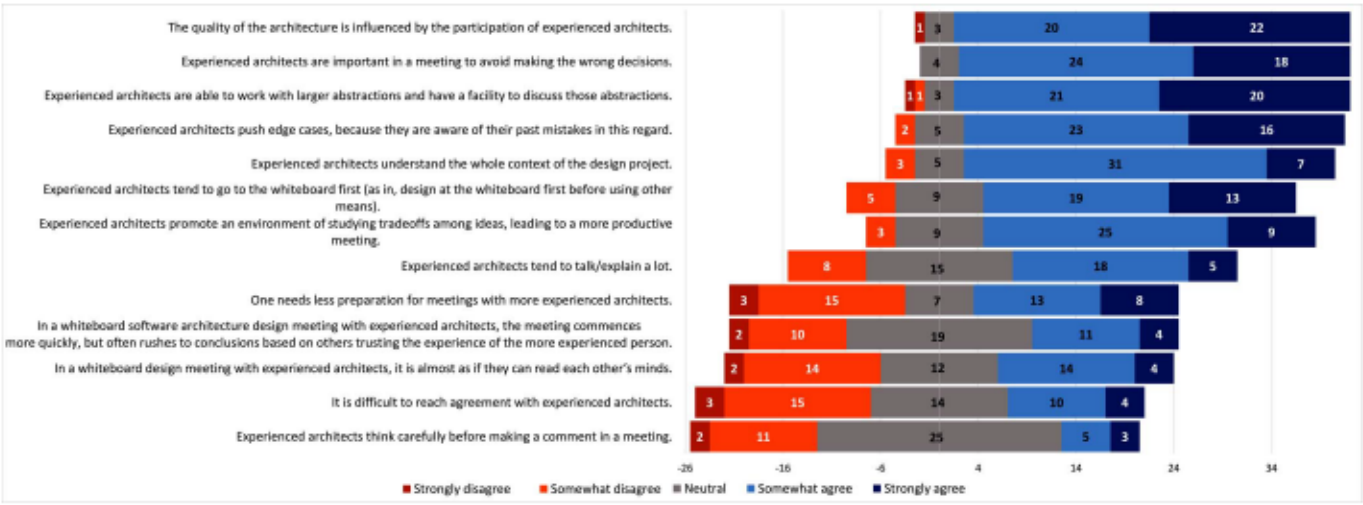

Fig. 3. Architects' perceptions on including experienced participants in whiteboard software architecture meetings.

neutral to this statement. The average Likert score for this statement is 4.35 (i.e., between "somewhat agree" and "strongly agree").

In addition to building an understanding of the perceptions of architects on the participation of novices, we equally sought to understand perceptions on the participation of experienced architects in the meetings. We identified thirteen such perceptions from the interviews and asked the surveyed software architects to rank these thirteen (Fig. 3). Again based on the level of agreeableness, the top five perceptions are: *the quality of the architecture is influenced by the participation of experienced architects* (average Likert score for this statement is 4.35, i.e., between "somewhat agree" and "strongly agree"), *experienced architects are important in a meeting to avoid making the wrong decisions* (4.3, between "somewhat agree" and "strongly agree"), *experienced architects are able to work with larger abstractions and have a facility to discuss those abstractions* (4.26, between "somewhat agree" and "strongly agree"), *experienced architects push edge cases, because they are aware of their past mistakes in this regard* (4.15, between "somewhat agree" and "strongly agree"), and *experienced architects understand the whole context of the design project* (3.91, between "neutral" and "somewhat agree"). The following are some comments from the interviewed software architects that highlight these aspects:

✓*SA 10: "The quality of the solution with more experienced people considers requirements that less experienced people will not consider. Then we will have a more stable and robust solution with more experienced people."* [the quality of the architecture is influenced by the participation of experienced architects]

✓*SA 7: "It's better because sometimes some decisions take a lot of work and if the decision is made wrong by a less experienced architect setting the course of a particular project in the next three, four weeks, we will lose a lot of time. So the participation of experienced architects at these meetings is imperative."* [experienced architects are important in a meeting to avoid making the wrong decisions]

✓*SA 18: "The conversation is not the same level. You have experienced architects it is almost that you are reading one another mind. So it is more diagrammed and less talked about, and when you have a discussion it is discussions that people go deeper or that end up becoming a proof of concept because they do not have a clear answer."* [in a whiteboard design meeting with experienced architects, it is almost as if they can read each other's minds]

In our survey, we also asked respondents whether they felt that overall it is important to include experienced architects in whiteboard software architecture design meetings. Among the 46 survey respondents, 37 respondents (80.43%) strongly agree that it is important to include experienced architects in the meeting, 6 (13.04%) somewhat agree, and 2 (4.35%) are neutral. The average Likert score for this statement is 4.7 (i.e., between "somewhat agree" and "strongly agree").

Observation 4

The top five perceptions on including experienced architects in whiteboard software architecture meetings are: the quality of the architecture is influenced by the participation of experienced architects; experienced architects are important in a meeting to avoid making the wrong decisions; experienced architects are able to work with larger abstractions and have a facility to discuss those abstractions; experienced architects push edge cases, because they are aware of their past mistakes in this regard; and experienced architects understand the whole context of the design project.

Beyond ranking the effects on whiteboard meetings of including experienced architects, we also asked the surveyed architects to select the five most valued aspects of experience that experienced architects should exhibit out of the ones identified in the discussion of Subsection IV.C.1. Table III shows the eighteen aspects, as ordered by frequency by which they were included in the top five.

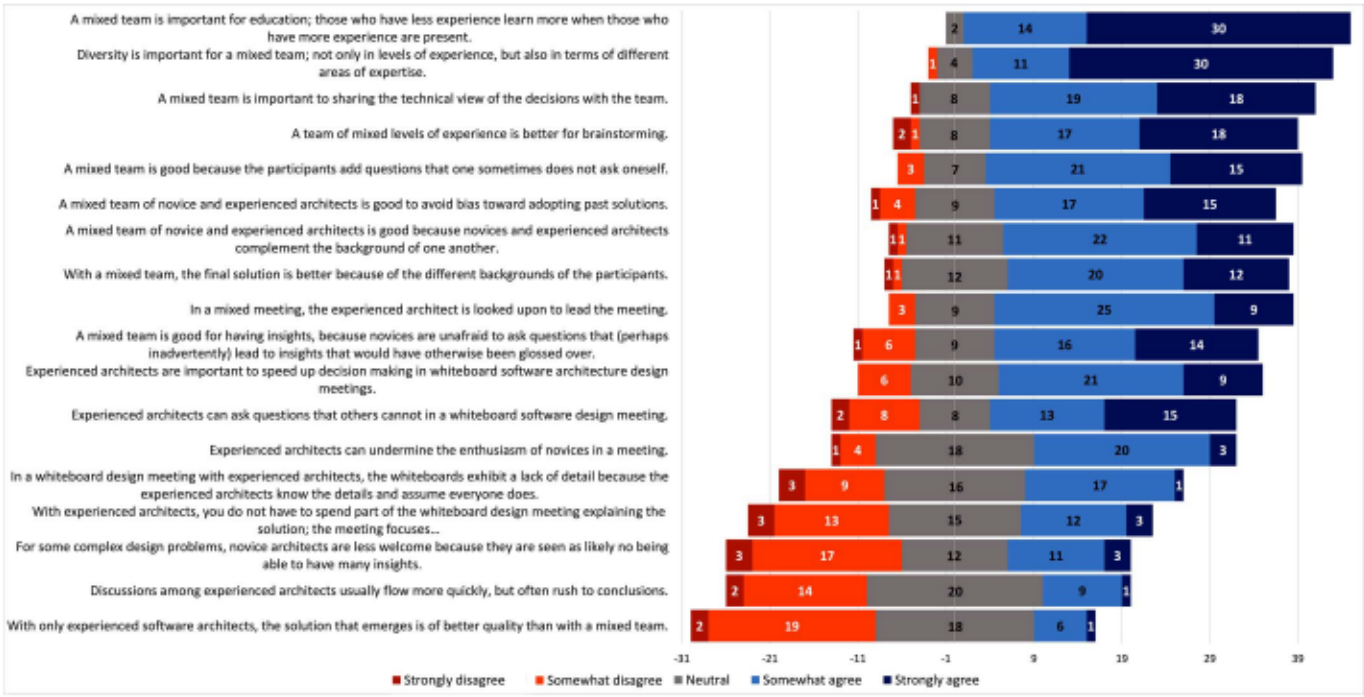

Fig. 4. Architects' perceptions on having a mixed team of participants in whiteboard software architecture meetings.

Observation 5

The top five most valued qualities for experienced architects participating in whiteboard software architecture meetings are: posses a foundation of architectural knowledge (patterns, methods, and tools); ability to see and analyze trade-offs; experience in having designed several architectures in the past; breadth of knowledge across domains, applications, and abstractions; and ability to both introduce ideas and serve as a sparring partner for them.

Because most whiteboard software architecture meetings involve a mix of novice and experienced participants, we also asked the interviewed software architects about their perception on mixed teams, which led to eighteen different perceptions that we asked the surveyed software architects to rank (see Fig. 4). The top five perceptions are: *a mixed team is important for education; those who have less experience learn more when those who have more experience are present* (average Likert score for this statement is 4.61, i.e., between "somewhat agree" and "strongly agree"), *diversity is important for a mixed team; not only in levels of experience, but also in terms of different areas of expertise* (4.52, between "somewhat agree" and "strongly agree"), *a mixed team is important to sharing the technical view of the decisions with the team* (4.15, between "somewhat agree" and "strongly agree"), *a team of mixed levels of experience is better for brainstorming* (4.04, between "somewhat agree" and "strongly agree"), and *a mixed team is good because the participants add questions that one sometimes does not ask oneself* (4.04, between "somewhat agree" and "strongly agree"). The following are some comments from the interviewed software architects that highlight these aspects:

✓*SA 9: "I think we should have people with various levels of experience, because those who have less experience learn more with those who have more experience. The discussion becomes more concentrated with experienced people, I think it is natural too, but it is important for everyone to participate*

TABLE III
EIGHTEEN MOST VALUED ASPECTS OF EXPERIENCE FOR EXPERIENCED ARCHITECTS

Aspect of Experience	Frequency
Posses a foundation of architectural knowledge (patterns, methods, and tools)	29
Ability to see and analyze trade-offs	25
Experience in having designed several architectures in the past	21
Breadth of knowledge across domains, applications, and abstractions	21
Ability to both introduce ideas and serve as a sparring partner for them	19
Ability to communicate with the team	19
Depth of knowledge in a particular area of specialty (domain, technology)	16
Ability to facilitate fruitful discussions	15
Ability to mentor others	11
Understand failure cases	10
Experience in having participated in many different projects	10
Ability to stop the team from going in the wrong direction	8
Understanding of the architecture of other existing systems	8
Many hours of hands-on software development	6
Awareness of technology trends	5
Ability to interact positively with the project manager	4
Ability to diffuse interpersonal situations	2
Knowledge of existing design decisions and rationale	1

so that everyone has the complete understanding of what is being discussed and everyone can learn how to do it, because the less experienced in the future will be those people who will lead these activities, so they need to participate from the beginning." [a mixed team is important for education, those who have less experience earn more when those who have more experience are present]

✓*SA 11: "I do not think the result comes out better only with experienced people. I think the outcome is best suited to diversity as long as everyone is involved and participating and have confidence in each other to participate. When there is such confidence and comfort, I prefer a more diverse environment, for even in the basic question insights arise, with people who think differently, better insights emerge, so I see more diversity. Diversity not only of experience, but also of background of areas. At our whiteboard meetings, the person who is project manager, who has a different background, a business analyst, a person of operation, participates. The best insights come from this mix."* [diversity is important for a mixed team, not only in levels of experience, but also in terms of different areas of expertise]

✓*SA 6: "When you are working with something that does not exist and something innovative, it is interesting to have a mix of people, even because you do not know what the roadmap will be or how the architecture will materialize, how will be the implementation itself, because a lot of architectural level, you can have assertion, define interfaces among subsystems, but at the time of going into the details, mainly, for integration between existing systems, there are many details that only arise when we do a more detailed design, and this kind of question, the novices are more willing to do it for lack of knowledge and this makes you think, explain, review the concepts and you even discovering what you do not know what you thought you knew and that enriches the discussion."* [a mixed team is good for having insights, because novices are unafraid to ask questions that (perhaps inadvertently) lead to insights that would have otherwise been glossed over]

In our survey, we also asked respondents whether it is important to include both novices and experienced software architects in whiteboard software architecture design meetings. Out of the 46 respondents, 27 strongly agree, 17 somewhat agree, and 1 was neutral with respect to this statement. The average Likert score for this statement is 4.5 (i.e., between "somewhat agree" and "strongly agree").

Observation 6

The top five perceptions on including novice and experienced participants in whiteboard software architecture meetings are: a mixed team is important for education; those who have less experience learn more when those who have more experience are present; diversity is important for a mixed team, not only in levels of experience, but also in terms of different areas of expertise; a mixed team is important to sharing the technical view of the decisions with the team; a team of mixed levels of experience is better for brainstorming; and a mixed team is good because the participants add questions that one sometimes does not ask oneself.

D. Documentation (RQ3)

Even the most suitably designed software architecture is useless if the people who need to use it do not know what it is, cannot understand it well enough to apply it, or misunderstand it and apply it incorrectly. All of the effort, analysis, hard work, and insightful design on the part of the architects will have been wasted. Thus, creating an architecture is not enough. It has to be communicated in a way that stakeholders can use it properly [48].

From the interviews, we learned that whether an architect documents their whiteboard software architecture meetings varies drastically, from never to always. We used the survey to better understand how often they do. Among the 46 survey respondents, 3 (6.53%) always document what happens during the meetings, whereas 21 (45.65%) document most of the time, and 11 (23.91%) only about half of the time. Ten others (21.74%) infrequently document what happens during the meeting and one (2.17%) does not document ever. The following are some comments from our software architects (SA) that highlight these aspects:

✓*SA 14: "We always take pictures of the whiteboard. Sometimes we have to delete the whiteboard, so we take a picture, then take it again, so sometimes we have three, four pictures of a single meeting. Usually this is broken. If it is a very complex flow, we digitize it in the sense of redoing that flow with some software, with some presentation or in the same flow draw itself. Sometimes we have to better specify what we have said, so we write a story and sometimes we have to break the cases and better document what each of those parts will be."*

✓*SA 8: "I do not particularly like documentation very much because it is very difficult to keep the documentation current with reality. It is very common we left an architecture meeting, go to a Wiki, we draw everything, we put text, images and I have particularly never seen this evolve in conjunction with the code. You get some of reference, initial, etc., but at some point this will be outdated. At some point people will no longer update that. It's very difficult for any change people update the document to reflect what's happening. So I am particularly against documentation: over documentation."*

1) Documentation Approaches: During the interviews, we also asked the software architects to describe what forms of documentation are used. From the answers, we identified fifteen different approaches, which we seeded into the survey to understand how many of the survey participants used each of the different approaches (see Table IV). The top five documentation approaches are: *photo(s) of the whiteboard*, *Wiki pages*, *notes taken during the meeting by one or more participants*, *notes produced/polished after the meeting*, and *photo(s) with additional notes*. The following are some comments from our software architects that highlight these choices, together with the documentation approaches:

✓*SA 24: "I never document in a formal way. So, many times, what is happening is that you do is you design things on the whiteboard and you make photos and as I said you put them either into a folder."* [photo(s) of the whiteboard]

TABLE IV
FIFTEEN APPROACHES USED TO DOCUMENT WHITEBOARD SOFTWARE ARCHITECTURE MEETINGS

Approach	Frequency
Photo (s) of the whiteboard	38
Wiki pages	32
Notes taken during the meeting by one or more participants	28
Notes produced/polished after the meeting	17
Photo(s) with additional notes	17
UML diagrams	15
Powerpoint slides	13
Informal record of decisions, alternatives, and rationale for choices	11
Informal record of decisions	10
User stories	9
Flow charts	8
We leave the whiteboard up ("Do not erase")	7
Architecture Decision Records (ADRs)	6
Use cases	6
Filling of Jira issues, stories or epics with decisions	1

✓*SA 15: "It usually depends on the type of discussion. If I want to validate just one quick idea, just to discuss one detail, usually we only take a photo of what is on the whiteboard. This is very common. So we put the photo on the Wiki directly, to show that it had the discussion. We try to document and to register, unless it is a tiny thing, we are going to join quickly and to discuss a thing of half an hour. But usually, we usually take a picture to remember what we discussed, but the general rule is to take that and write a page or two. Not documenting is an exception."* [Wiki pages]

We note that these approaches range from capturing the results 'as is' (e.g., photo(s) of the whiteboard, leave the content of the whiteboard up), through some filtering and processing (e.g., notes produced/polished after the meeting, informal record of decisions), to what can be a more elaborate pass during which further refinement can take place by the person documenting the results of the meeting (e.g., UML diagrams, flow charts). In the latter case, it is not uncommon for the results to undergo further scrutiny, either in a separate meeting where the more formal document is reviewed or asynchronously through commenting on a shared document. Any refinements made at that point represent the start of what can be further downstream changes (see Section IV-E).

We also note that the approach followed for documenting in some cases clearly was connected to the content of the whiteboard (e.g., using UML to document a component diagram that was created on the whiteboard, using a flow diagram to document a distributed protocol that was explored on the whiteboard). In other cases, however, participants indicated using a single or a few documentation approaches indiscriminately, with, for instance, photographs of the whiteboard or a note taker being responsible for sending an outcome and discussion summary via Slack or e-mail the standard approach regardless of the kind of content discussed at the whiteboard. As such, there is no strong correlation, with different approaches used for similar kinds of situations. That said, specific approaches such as using user stories or ADRs only are applicable in certain situations and it is clear that more generic approaches are used most often.

TABLE V
TWELVE REASONS WHY SOFTWARE ARCHITECTS CHOOSE TO DOCUMENT WHITEBOARD SOFTWARE ARCHITECTURE MEETINGS

Reason	Frequency
Serve as a starting point for follow-up discussion in future meetings	38
Retain as evidence for later of the decisions that were made	33
Participants forget	30
Communicate the outcomes to others on the project	27
Different participants have different beliefs regarding the outcomes of a meeting; documenting helps disambiguate	26
Use later to educate new people on the project	14
Validate in detail whether the design ideas indeed can work as intended	11
Train the team on the design	8
Enable reuse of the design ideas in other design projects	8
Present a preliminary solution to the customer	5
Participants sometimes second-guess what they did	5
Include as part of the design that we are handing off to the customer (so they can do the implementation work)	3

Observation 7

A majority (76.09%) of software architects always or most of the time document what happens in whiteboard software architecture meetings. Photo(s) of the whiteboard; Wiki pages; notes taken during the meeting by one or more participants; notes produced/polished after the meeting, and photo(s) with additional notes are the five most common documentation approaches used by the software architects to do so.

2) Reasons to Document the Meetings: Complementing whether software architects document what happens in whiteboard software architecture meetings and how they do so, we also sought to understand their primary reasons for investing time and effort in documenting. From the interview sessions, we identified twelve reasons, from which we asked each survey respondent to select the five most important to them.

As shown in Table V, the five reasons that were most frequently selected are: *serve as a starting point for follow-up discussion in future meetings*; *retain as evidence for later of the decisions that were made*; *participants forget*; *communicate the outcomes to others on the project*; and *different participants have different beliefs regarding the outcomes of a meeting; documenting helps disambiguate*. The following are some comments from the software architects that exemplify some of these aspects, together with the reasons to document the meetings:

✓*SA 13: "They are of course used because you take this as a starting point to continue the work. So, they are essential to the participants and sometimes also to other interested people who get the results by email. And later on they are looked at, as I said the meetings are just the starting point to discuss part of a solution or to provide some analysis that you can support next discussion with data."* [serve as a starting point for follow-up discussion in future meetings]

✓*SA 14: "The meeting cannot stay in people's head, it needs to be documented in some way. Someone needs to*

commit to transferring that knowledge that has been generated, sometimes on the whiteboard itself or at other times in discussions, someone needs to be documenting this so that it becomes a task." [participants forget]

✓*SA 9: "Every time we have a new person on the team, we recommend [ed: them] to see the architecture documents and present doubts or explain [ed: their] understanding of the architecture. It is to make sure [ed: they] understood what we designed. So, the documents are used as reference, whenever someone has a doubt about a piece or a component designed, we recommend the person to go back to the architecture document."* [use later to educate new people on the project]

Observation 8

The top five reasons to document whiteboard software architecture meetings are: serve as a starting point for follow-up discussion in future meetings; retain as evidence for later of the decisions that were made; participants forget what was discussed; communicate the outcomes to others on the project; and different participants have different beliefs regarding the outcomes of a meeting/documenting helps disambiguate.

E. Downstream Changes (RQ4)

Various studies document what happens at the whiteboard (e.g., [21], [22], [23], [26], [28], [29], [31]). Other studies examine architectural decay and erosion: what happens to an architecture once it has been realized in an implementation and changes are necessary to the code base that have architectural implications (e.g., [25], [49], [50]). What has not been studied yet, however, is what happens in between: once a whiteboard meeting concludes, what kinds of downstream changes are necessary to be made to the architecture as it is being rolled out and refined? As mentioned in Section IV-D, such changes may already emerge just from the act of documenting what happened at the whiteboard (an architect more formally documenting decisions after a meeting may realize that some aspect of the architecture as designed has a flaw and decide to address the flaw on the spot since it is not too onerous to do so) or may happen later (for example, when the architect debriefs the team on the decisions made and in the process of explaining realizes an issue, or when a developer responsible for making some changes in the midst of making those changes encounters a problem in how the envisioned architectural changes are incompatible with some aspect of the current codebase).

In our interviews, we asked architects about what kinds of changes they have witnessed being made to the architectural designs and plans they created in whiteboard software architecture meetings. Table VI shows the ten different aspects that were recounted by the software architects, in order of how often they were mentioned by the surveyed software architects. The top five aspects that change downstream are: *interface of major components in the architecture*; *implementation details*; *a small handful of components in the architecture*; *detailed modules inside architectural components*; and *database schema*.

TABLE VI
TEN ASPECTS THAT CHANGE DOWNSTREAM FROM WHITEBOARD ARCHITECTURE MEETINGS

Aspect	Frequency
Interface of major components in the architecture	34
Implementation details	28
A small handful of components in the architecture	24
Detailed modules inside architectural components	21
Database schema	20
Driving scenarios	19
Format of messages connecting various parts of the architecture	13
Overall architectural solution	13
Key algorithms	7
Change of proposed implementation (time or scope)	1

The following are some comments from the interviewed software architects that highlight these aspects, together with the kinds of changes:

✓*SA 9: "I think when you do not think of all the details. I remember that we were once designing part of the KNoT device protocol and we thought and wrote it on the whiteboard, but we did not think at all. So we had to add more things we had not thought about when we were designing. They are things like that. They are parts of the implementation that you do not think about when you're on the whiteboard, but then you realize you need it in implementation. It is which I think usually changes."* [implementation details]

✓*SA 8: "What I see to change is usually the format of the messages. We define the messages with certain attributes and then we see that something is missing and we have to adjust with more or less attributes due to performance, for example, the payload got too big and we need to decrease the payload due to performance or we need to send more attributes because the domain or functional requirement changed on the other side and we will need to consume more information than what was originally intended. It is something that usually changes a lot over the life of an application. It is something I see happen frequently, we define that the message will be this and for some reason we have to change that message or increase or decrease for some reason."* [format of messages connecting various parts of the architecture]

Observation 9

The top five aspects that change downstream from whiteboard architecture meetings are: interface of major components in the architecture; implementation details; a small handful of components in the architecture; detailed modules inside architectural components; and database schema.

1) Rationale for Changes: After identifying the aspects that change downstream, a natural step was to understand the rationale for the changes. From the interviews with the software architects we identified seventeen reasons to make changes and then asked the surveyed software architects to mark the five that they experienced most frequently. Table VII shows the seventeen reasons to make changes together with their frequency. The top five reasons for change are: *certain aspects of the solution are over-simplified and turn out to be more complex*;

TABLE VII
SEVENTEEN REASONS TO MAKE CHANGES DOWNSTREAM FROM WHITEBOARD ARCHITECTURE MEETINGS

Reason	Frequency
Certain aspects of the solution were over-simplified and turn out to be more complex	28
We discovered a better solution than the original we devised at the whiteboard	23
Multi-dimensionality of the problem – qualities that were not considered (or merely lightly considered) during the whiteboard meeting are negatively affected by the planned solution	23
The project is Agile, and thus had to respond to new circumstances	23
Performance	19
Customer requirements changed midstream	17
Technology/platform limitations	17
Team made false assumptions	14
Difficulty in mapping the high-level solution to actual code	9
Scalability	9
Certain predictions of how the architecture would behave did not hold up	8
Reliability	7
Lack of having documented what we did at the whiteboard	3
Team misunderstood the architectural design	3
Social problems with the team	2
All the above at different times, it is very context dependent	1
The original meeting was not conducted very well and thus not effective	1

TABLE VIII
SIXTEEN ASPECTS MISSING FROM WHITEBOARD SOFTWARE ARCHITECTURE MEETINGS THAT COULD HAVE IMPROVED THE OUTCOMES

Aspect Missing	Frequency
Sufficient information about the problem to design the solution	29
Understanding of the relative priority of various design considerations	24
Metrics that delineate 'success' of the architectural design	19
Validity of assumptions about decisions, as to whether they hold up at implementation time	19
Details about the envisioned implementation	19
Certain requirements	18
Agenda for the meeting	14
Details about the current implementation	13
Test cases governing the architectural design	9
Dependencies among the various whiteboard sketches	9
Context diagram	9
Interfaces among the components	8
Structure of the messages exchanged by the components	2
An overview of the project	2
Clear problem to be solved	1
Clear next steps and assigned responsibilities	1

architects discover a better solution than the original at the whiteboard; *multi-dimensionality of the problem – qualities that were not considered (or merely lightly considered) during the whiteboard meeting are negatively affected by the planned solution*; *the project is Agile, and thus had to respond to new circumstances*; and *performance*. The following are some excerpts from the interviews that highlight a pair of these aspects, together with the rationale for the changes:

✓*SA 19: "Good question. What sometimes changes, in fact, does not change so much because what we do on the whiteboard is very macro, it is an overview, it does not go as far as detail, but what sometimes changes is that sometimes you think of using a technology, a framework, and when you go to Google, Stack overflow, or talking to a colleague, you discover another technology is better. For example, a text search with Solr, and you find out that everyone is using Elasticsearch, and you think: let's use Elasticsearch. So you find you have a more interesting alternative."* [we discovered a better solution than the original we devised at the whiteboard]

✓*SA 13: "So, sometimes you get to know more details about some aspects so, we have multi-dimensional problems usually. So, this is embedded software which is variable, being embedded it is also resource-constrained, it is real-time running on the multicore processors and there are also some other architectural qualities, maintainability and so on that need to be addressed. So, frequently the whiteboards discussions concentrate on some specific quality or on some specific problems. Only then you realize: okay, if we do it that way then maybe a third or fourth architectural quality will suffer."* [multi-dimensionality of the problem – qualities that were not considered (or merely lightly considered) during the whiteboard meeting are negatively affected by the planned solution]

Observation 10

The top five reasons to make changes from sketches to implementation are: certain aspects of the solution are over-simplified and turns out to be more complex; architects discover a better solution than the original at the whiteboard; multi-dimensionality of the problem – qualities that were not considered (or merely lightly considered) during the whiteboard meeting are negatively affected by the planned solution; the project is Agile, and thus had to respond to new circumstances; and performance.

2) Missing Meeting Aspects as Potential Causes: One somewhat unexpected theme that emerged from the interviews is that the architects talked about 'what could have been': aspects of whiteboard software architecture meetings and how they were conducted that, had they been done differently, could perhaps have avoided future changes being necessary. Based on the interviews, we identified sixteen such aspects from which, once again, each surveyed software architect could tag five as what for them were most frequently 'missing aspects': aspects that had they been incorporated better may have improved prior meeting outcomes.

The top five aspects that resulted were: *sufficient information about the problem to design the solution*; *understanding of the relative priority of various design considerations*; *metrics that delineate 'success' of the architectural design*; *validity of assumptions about decisions, as to whether they hold up at implementation time*; and *details about the envisioned implementation*. Table VIII shows the sixteen aspects

in order of frequency in which they were tagged by the surveyed architects. The following are some comments from the software architects:

✗*SA 12: "That's a good question. I think it's the depth of the impact of architecture on the solution. Sometimes this happens, for example, we integrate a video platform of a television channel with 110 thousand videos approximately. We drew the entire model while we did not have the total volume of videos from the customer. Sometimes we have little knowledge of the whole context the solution requires and this has a lot of impact on implementation. It makes a very big difference you implement a solution for 10,000 and another for 110 thousand videos with 3T of storage. What is missing sometimes on the whiteboard is enough information to get the solution."* [sufficient information about the problem to design the solution]

✗*SA 8: "Success criteria and evaluation criteria. Whether it is good or not and how we are going to measure things. As we measure performance, uptime, fault tolerance, this kind of thing is invariably missing."* [metrics that delineate 'success' of the architectural design]

Note how these factors represent a mix: some concern having additional information at hand, some the conduct of the meeting itself, some additional angles of the design that they wished they had worked out in more detail, and some the criteria by which the architecture eventually would be judged.

Observation 11

The top five aspects missing from the whiteboard software architecture discussions are: sufficient information about the problem to design the solution; understanding of the relative priority of various design considerations; metrics that delineate 'success' of the architectural design; validity of assumptions about decisions, as to whether they hold up at implementation time; and details about the envisioned implementation.

Previous research has shown that different kinds of media are used for architecture design. Beyond the whiteboards, these media may include scrap paper to informally sketch and model, but also software tools like Photoshop and Powerpoint [32]. The past decade also has seen the emergence of a new crop of tools, such as the Microsoft Surface and other devices which enable touch based design, as well as cloud-based, remote collaboration oriented whiteboard tools such as Gliffy[4] and Miro[5]. These kinds of tools offer new opportunities, both in terms of how teams work together and who is brought into meetings (e.g., remote participation is much easier so meetings can be more inclusive) and in terms of moving outcomes downstream (e.g., many of these tools have export capabilities, some are tightly integrated with other tools such as Wikis and task managers). Thus, it is important to understand software architects' perceptions about these digital tools and the impact on their activities. We used the survey to do so.

[4]https://www.gliffy.com/
[5]https://miro.com/

V. Discussion

Our study takes a look at software architecture meetings at the whiteboard, a setting that to date has not been principally examined. Through a combined interview and survey study, we focus on the reasons why architects go to the whiteboard, the impact of the experience of the participants in the meetings, how meeting outcomes are documented, and what kinds of downstream changes are necessary to the architecture as first outlined in the whiteboard meeting. Some of our findings align with findings that have been made about whiteboard meetings more broadly. As one example, the fact that architects go to the whiteboard for a variety of reasons aligns with programmers using the whiteboard for many different reasons [21]. Other findings align with the literature on expertise and the roles that experts and novices play in creative endeavors by, for instance, confirming that the questions asked by novices can cause experts to reflect more deeply and as a result reassess assumptions that hitherto they had not considered important to discuss [51]. Yet other findings, however, contribute novel insights as related to the unique setting of our study: whiteboard software architecture meetings. Below, we first discuss the primary takeaways as anchored in the results of Section IV and then discuss the collective implications for research, practice, and education.

A. Primary Takeaways

Underneath the eleven observations that we already articulated in the above are some important takeaways that characterize whiteboard software architecture meetings. Below, we introduce and contextualize each takeaway.

Takeaway 1: Software architecture whiteboard meetings concern high-level work, blend understanding the problem with creative brainstorming and idea generation toward a solution, and are seen as an important venue for educating others. Across all of the reasons software architects mention for going to the whiteboard (Fig. 1 and Section IV-B), most paint a picture of the work performed in the meetings being at a high level and oriented toward problem setting and figuring out an overall direction for the architecture or architectural change being discussed. This is consistent with prior observations regarding the nature of software architecture work more broadly (e.g., [52], [53]), though our results delineate a nuance in that whiteboard work represents a subset that in general stops short of specifying an architecture in full detail. Creativity and brainstorming are an important part of the meetings, which is consistent with prior work regarding design and other work at the whiteboard (e.g., [21], [23]), with the novel finding that a major secondary purpose of the whiteboard meetings is educating colleagues either through organizing a whiteboard meeting specifically for communicating already made decisions or more indirectly through others' presence and participation.

Takeaway 2: Software architects consider it important that whiteboard meetings involve both experienced and more novice participants, with the expectation that experienced participants exhibit strong technical skills garnered through exposure to prior architecture design, a broad range of existing systems and domains, failure, and

more, as blended with more non-technical skills concerning communication and teamwork to ensure effective ideation, discussion, and consideration of tradeoffs. Since whiteboard architecture meetings are ultimately solution focused, it is important that the team that is assembled at the whiteboard has the right technical expertise to produce such solutions. Prior expertise is seen as a strongly contributing factor to be able to do so successfully, though it is interesting to observe that such experience is not solely with the particular system at hand. Instead, broad architectural knowledge and experience appears as important as deep domain knowledge, and both types of expertise are explicitly sought to be brought together in whiteboard software architecture meetings. An additional novel finding is that the architects we interviewed and surveyed had a strong preference for including novice participants in the meetings. While they acknowledged drawbacks to doing so, in terms of making meetings longer or having to explain things they do not have to explain with just experienced architects, the benefits in terms of sometimes unearthing wrong assumptions because of having to respond to questions from novices or the generation of a broader range of ideas when brainstorming, appear to outweigh the drawbacks. Finally, the inclusion of novices is also seen as essential in educating them, both in the architecture and decisions being made as well as in training them as budding architects.

Takeaway 3: Whiteboard software architecture meetings are documented to serve as a starting point for future activities and to remember what transpired, with the mechanisms by which the meetings are documented varying widely. Few architectures are designed in a single meeting; most are the result of a series of activities [54], [55], [56] of which the whiteboard meeting is but one. It is unsurprising, then, to find that one reason architects capture whiteboard meetings is to provide a starting point for future activities, be it follow-up discussion, communicating outcomes to others for detailed design or implementation, or validating the envisioned architecture in detail. The full compendium of reasons why they capture the meetings, however, also includes reasons that relate to remembering important aspects of the meeting outcomes that might later be forgotten, misremembered, or remembered differently among participants. Meeting outcomes are primarily documented informally by taking photos, adding content to a Wiki, or having one of more participants take notes (that may or may not be polished after the meeting and may or may not accompany photos). On occasion, outcomes are documented more formally, e.g., as a UML diagram. Overall, these findings present a tenuous picture. On the one hand, architects recognize the importance of generating documentation, presumably because of the importance of the decisions being made in whiteboard software architecture meetings. At the same time, informal documentation approaches are known to lead to potential problems [57]. We speculate that the memory of the meeting participants often still is relied upon most when knowledge is passed on to later activities.

Takeaway 4: Downstream changes to the architecture as designed in whiteboard software architecture meetings stem from a variety of reasons, including changing external circumstances and, most prevalent, aspects of the solution being oversimplified when they should not have been; the architects recognize that a frequent underlying cause is that certain relevant information was not brought into the discussion or simply was not yet available at the time. Our study is a first that sheds light on what changes are typically necessary when an architecture as envisioned at the whiteboard is refined in future activities; these changes are different from architectural erosion and decay in that the changes occur while the architecture is still being conceptualized. Changes needed range from implementation details and format of messages to having to reconsider the entire architectural solution, driving scenarios, and interfaces among major components. Architects are able to pinpoint 'what went wrong', with—per Table VII—oversimplification of the solution and a lack of understanding of the full complexities of the problem (e.g., certain qualities were not considered, performance, technology/platform limitations) as two key reasons for why they have to make downstream changes. A novel finding is that architects actually are aware of what kind of information should be brought into the meetings to help them avoid having to make such later changes (Table VIII), but, given that downstream changes continue to be necessary, apparently cannot do so on a consistent basis.

B. Implications for Research

Our results give ride to a number of research directions that we believe are important to pursue next, the primary ones of which we introduce in the below.

Research Implication 1: Empirically validate the varied findings concerning whiteboard architecture meetings benefiting from a mix of experienced and novice participants. The perceptions of the architects as discussed in Section IV-C on what they believe the effects of different levels of experience are on how the meetings proceed remain perceptions. The fact that many of the perceptions receive a significant amount of agreement from the surveyed architects implies it is likely that many of these perceptions are largely accurate. At the same time, it is important to verify these perceptions with rigorous studies of the impact of the mix of experience in team composition on both how whiteboard software architecture meetings proceed as well as their eventual outcomes. For instance, one of the software architects observed: *"The quality of the solution with more experienced people considers requirements that less experienced people will not consider. Then we will have a more stable and robust solution with more experienced people."* Yet, as we already mentioned, it is also believed that novices can cause experienced architects to reconsider aspects of their design because of seemingly ignorant' questions, which equally can impact the resulting quality. Exactly where the balance lies will need to be studied carefully, perhaps along the lines of the experiments of [58], [59], [60], [61].

Research Implication 2: Study whiteboard software architecture meeting dynamics. The behavioral and psychological aspects of whiteboard software architecture meetings

should also be further investigated. Our findings are not unanimous in this regard. Perceptions such as "novice participants may be afraid to speak", "difficulties reaching agreement with merely experienced people", "novices can be inflexible", and "needing less preparation for meetings with just experienced architects" are mostly disagreed with by the surveyed architects, but some of them actually agreed with these statements. Even the perception that "the quality of the architecture is influenced by the participation of experienced architects" or "a team with mixed participants is better for brainstorming", which had near-unanimous support, see some surveyed architects strongly disagreeing. Understanding the meeting dynamics of varying groups may be able to uncover the differing conditions and practices that lead to such different perceptions. *"Another advice has to do with facilitation as well. It is making sure everyone in the meeting is heard. Sometimes we have more talkative and less talkative people and sometimes we have opinions that are left out because some people are more shy or not so vocal. So to have an effective meeting, facilitation is a crucial point."* Studies examining meeting conduct and people interactions exist in a more general sense (e.g., [62], [63]), but the domain of software and particularly software architecture has not been studied to date in that regard. With software exhibiting unique characteristics and challenges when it comes to meetings at the whiteboard, observational studies considering these aspects are welcome.

Research Implication 3: Revisit documentation techniques. In the context of informal documentation approaches being most common mentioned, together with the recognized failure of many proposed design rationale techniques to be adopted in practice [64], we consider it important to re-engage in research that seeks to on the one hand study where and when current documentation approaches succeed and fail, and on the other hand in new techniques for assisting architects in capturing whiteboard meetings. Interestingly, *lack of having documented what we did at the whiteboard* is mentioned by just three survey participants, which might indicate that current informal techniques are less of a problem than it often is portrayed to be (e.g., [55], [57], [64]), especially when contextualized by the many other reasons listed in Table VII as to why downstream changes are needed. Intuitively, the architects know they need to capture what was being discussed, but scant literature exists that shows the benefits of doing so: when is this info used, by whom, how is it making a difference, and what happens when the information is not available? Such studies could improve the understanding as to why one should capture which meeting outcomes in what concrete form.

Complementarily, developing improved methods of capturing meetings is an important avenue for future research. The emergent use of Architectural Decision Records (ADRs) presents an interesting middle ground in being lightweight, yet more structured than many of the unstructured techniques quoted in Table IV. The use of ADRs has recently gained some traction (six out of 46 surveyed said they have used ADRs, conform Table IV and the literature is also reporting on the beneficial role of ADRs [65]), One of the architects commented: *"Lately, we are experimenting with a technique called ADR, a template that we put in archives of the repository we are developing that documents the decisions. Then you open a PR file, with that decision, someone approves immediately and gets that decision. In general, it is a very simple and short document, sometimes it does not reach half a page of a document, but we record a decision that we want to record and return to it when we are making other decisions."*

Research Implication 4: Hybrid and remote. Although our study focused on collocated whiteboard meetings, we note that several architects brought up hybrid and remote meetings in the interviews. While we ignored those parts in our analysis, the interviewees understood the realities of today's post COVID-19 world in which hybrid and remote work is persisting and traditional whiteboard meetings must be adapted to include remote team members. *"We've got hugely distributed teams so face to face meetings are becoming less and less important. In fact, I've been working almost entirely remotely with people so we kind of we do sometimes have we can make a virtual whiteboard at design meetings but you know I think the tools for that are fairly primitive now, right."* A great many studies are emerging at this time surrounding the topic of remote and hybrid meetings (e.g., [66], [67]), including some emerging work on maintenance design by an architecture team [68], yet a focus on the creative and design aspects of whiteboard software architecture meetings remains absent. It is important to follow up this study with a study that examines whether the same perceptions remain, as well as what other perceptions emerge due to the different physical setting and different human behaviors arising in this setting.

Research Implication 5: Tools. Beyond the traditional physical whiteboard with pens and an eraser, which still continues to be used often for in-person meetings, many tools have been developed that provide a virtual whiteboard experience (e.g., Miro[6], Jamboard[7], Mural[8], ConceptBoard[9]). Particularly over the past few years, these tools have become increasingly popular and have seen a significant expansion in the types of features they include. While their primary purpose is to enable a team of remote participants to collaborate, a virtual whiteboard offers opportunities for additional functionality that could assist architects in whiteboard software architecture meetings. In several ways, they already do: they integrate advanced notations, support a variety of design techniques (e.g., mindmapping), and, because they are electronic, content persists. That said, from the interviews and the survey, room for improvement exists. *"We cannot search it, we cannot verify what was decided at the previous meeting for the which is being decided now, we cannot compare the decisions and know that you've been wrong for three months and that three-month error could be recorded somewhere. So I think the transition for the digital should be smarter, something like: "look, you are trying to*

[6]https://miro.com/
[7]https://workspace.google.com/products/jamboard/
[8]https://www.mural.co/
[9]https://conceptboard.com/

write a structure that some time ago you have already defined, you do not want to reuse what you did? Or you're trying to make an implementation much like another system that did the same thing in a particular architecture sharing repository, let's say." We make two observations about this desire and particular comment. First, it points to needing better facilities to search for past content. Second, we note that part of what is being asked for already exists: current virtual whiteboards can turn hand-drawn sketches into more formal diagrams and representations that can then be exported to various other tools. The comment, however, seeks a much deeper integration, one in which such more formal documents and even prior sketches are fed back into the virtual whiteboard experience to more deeply assist the architects at work. Comments by several other architects similarly highlighted the need for not just supporting designing in the right notation, but to actually offer more 'smart support' for the activities at hand.

Separately, we believe virtual whiteboards could form the basis for smart, semi-automatic note taking. Tools such as otter.ai[10] already can auto-transcribe meetings and even provide rudimentary summaries. Combining such functionality with human guidance could help streamline the note taking process significantly and help focus the capture on those pieces of information that are mots likely to be needed in future (as explored, for instance, in KnoCap [69]). Perhaps an even more important function could be for virtual whiteboards to automatically deliver relevant information to the architects as they are designing. Since it is possible to "listen in" on conversations, potentially relevant information that was captured in the past could be non-intrusively suggested to the architects as being relevant, with the option for them to ignore such information or actually bring it up in more detail seamlessly in the meeting.

C. Implications for Practice

From our study, several important suggestions arise for practicing software architects and how they choose to conduct and engage in whiteboard software architecture meetings.

Practice Implication 1: Involve the right mix of participants. While this sounds in some ways too straightforward and is perhaps even redundant advice, since architects typically do consider whom they invite to the meetings and why, three dimensions stand out to which they should pay particular attention: experience, different perspectives, and relevant expertise. In terms of experience, the architects that we studied strongly feel that mixed levels of experience should be brought into the room, from highly experienced architects to much more novice architects. Each group challenges the other, causing broader discussions to take place that both consider aspects of the architecture that otherwise would not be considered and teach the novices how to become better architects through their participation (a key trait of experienced software designers is continuous learning about new technologies and other types of systems [70]).

[10]https://otter.ai

Beyond mixed levels of experience, including meeting participants who bring different perspectives to the discussion is crucial: *"What also helps when you have 2–4 participants is that they should have different perspectives, so they come from different organizational background or have a different expertise focus. You cannot know everything as a single person and if you get the second person which is similar to you from the profile it does not double the knowledge. But if you have a few people with different perspectives then you shed light from different directions on the problem and usually someone has a different perspective and sees other aspects of the problem which you could not come with because you do not even know that such thing exists"*.

Complementing experience and perspective is the importance of including people who have the relevant expertise: *"Another aspect is to bring the right people to the meeting. I have seen meetings that were not effective because we did not have the right people at the meeting. Let's discuss deploy, containerization, but no one knows enough of Docker to talk about it, does not know what the possibilities are, etc., so bringing people who know how to talk about it is important to get the findings faster"*. It still happens that meetings are conducted that fail these inclusion criteria.

Practice Implication 2: Promote psychological safety for meeting participants. *"Promote psychological safety, that is, psychological security for people to express opinions, so they do not feel frightened. When you are going to make a comment, which is a complete bullshit, that's fine, this should not have a consequence, it should not be mocking an opinion of a person who is sincere and is willing to contribute to the meeting. So the person leading the meeting needs to worry about all of these aspects so he can extract the most value from it"*. The importance of such psychological safety is well-known in the literature on how to conduct high-quality meetings (e.g., [71], [72]), but it is an important reminder for architects to recognize that one of their roles in these meetings is to create a welcoming and open environment for discussion.

An important element of promoting psychological safety is to conduct high-quality meeting, toward which the architects recognized a number of strategies, ranging from making sure that everyone is heard (*"Sometimes we have opinions that are left out because some people are more shy or not so vocal. So to have an effective meeting, facilitation is a crucial point."*), to defining and publicizing an agenda well before the meeting (*"I think that it is very important to have a meeting agenda. Sometimes it happens the meeting gets away from the topic and what we do is to set another meeting for the new, another topic. We try to stay focused on the problem that we have on our hands and use the time exactly for that."*), to sharing relevant materials beforehand so that meeting time can be spent constructively considering materials that have been read by the participants before the meeting starts rather than actually reading the materials on the spot, to including a facilitator (*"Regarding agreeing and making the meeting effective, it has a bit of facilitation as well. If facilitation is active, we can reach conclusions faster. Usually we are discussing a*

diagram or some proposals and we have opposing opinions, different proposals, and sometimes the quickest conclusion is: let's test both. And there must be a maturity in the facilitation to reach that consensus quickly."). In many ways, whiteboard software architecture meetings are just another type of meeting, so it is not surprising that these kinds of general lessons also apply here. We do note the particular importance of sharing relevant materials beforehand. From Table VIII, it is clear that a significant problem in these meetings is a lack of critical information, with the top four being insufficient information of the problem, knowledge of the relatively priority of various design considerations, metrics that delineate success, and an understanding of the assumptions being made and how valid those assumptions are. This is all information that an architect could and should prepare beforehand, so all participants have a shared sense coming into the meeting. Moreover, it avoids that sharing such information in the meeting is seen as a distraction from making progress when a meeting participant asks for clarification or, worse yet, is simply forgotten to be shared in the first place.

Practice Implication 3: Consider potential downstream changes early. While some set of downstream changes are relatively innocuous (e.g., adding more detail to an architecture), others cause rework (e.g., driving scenarios that change, overall architecture being reconsidered). The architects understand the typical reasons as to why the architectures change downstream (Table VII). Some of those reasons have to do with the process being followed (e.g., lack of having documented what we did at the whiteboard, social problems with the team), others with unpredictable changing circumstances (e.g., customer requirements changed midstream, the project is Agile and thus had to respond to new circumstances), and yet others concern the architects not considering the problem and solution to the fullest (e.g., certain aspects of the solution were oversimplified, multi-dimensionality of the problem). One approach to address dealing with potential future downstream changes early is for architects to have a checklist of the typical causes, and invite participants to brainstorm and discuss for each of the causes whether it is applicable to the situation, how much information the team has to substantiate the likelihood, what information could help further clarify, and if any action can be taken now (or shortly after the meeting) to minimize the impact. Especially in light of the sixteen aspects typically missing from whiteboard meetings as listed in Table VIII, a majority of which concern a lack of information, foregrounding the reasons in the meeting and addressing what information is missing can potentially help stem disruptive future changes.

Practice Implication 4: Consider when documentation is necessary and in what form. Documentation appears not necessary all the time, with, for instance, architects sometimes simply relying on their memory of the meeting to instigate downstream activities. As such, dogmatically documenting each meeting and its outcomes through extensive note taking possibly with more formal diagrams attached such as UML or flow charts would represent wasted effort. On the other hand, there clearly are situations where documentation is important to have (Table V). Architects are mindful of the effort involved, however: *"Because it will change and it is very difficult to maintain the consistency of what is in the code with this abstraction which is its architecture. It is to spend bullet with deceased dead. It should be used as a reference, as I said: you say when we started was like that and we changed because of these aspects, it is much more static you document the principles that were used in your architecture than the architecture draw."* Within this context, then, it is not surprising that more lightweight methods are mentioned most frequently as being used (Table IV), although some architects use certain methods always and others do so more selectively. In terms of concrete advice, it might be worthwhile to focus the documentation aspects on those parts of the architecture that are likely to change (conform Table VI). This is a somewhat counter-intuitive idea, as normally one tends to concentrate on documenting those parts of the architecture that are well understood and firm. Yet, documenting those parts that are more likely to change has the potential benefit of creating artifacts that can be discussed earlier (and thus with less potential cost in terms of already implemented code) and that, by virtue of being marked as tentative can invite such further discussion. Moreover, when changes are needed, they can be done with an explicit representation of what was discussed in the past, which represents an important starting point and avoids having to re-invent the wheel or re-constructing the prior discussion.

D. Implications for Education

The insights we garnered provide fertile ground for how students are educated in the topic of software architecture as well.

Education Implication 1: Conducting whiteboard meetings. Beyond the need to cover architecture as a separate topic (which many programs do only peripherally so, although exceptions exist [73], [74]), perhaps the most important factor is to teach the importance of architecture meetings at the whiteboard: what is typically discussed, how to conduct them, what kinds of perspectives should be brought to the discussion, the role of sketches in supporting the discussion taking place, and how to take those sketches into further development activities, and more. Soft skills are an under-taught aspect of software engineering. With software architecture serving a crucial role in the development of systems, and with the collaborative work at the whiteboard a key element in their design, the need for new approaches to teach these topics is high. Existing courses on how to design software (e.g., [75], [76], [77]) as well as software maintenance (e.g., [78], [79]) might provide both inspiration and serve as potential starting points in this regard.

Education Implication 2: Architectural experience and practice. Beyond traditional software engineering courses in degree programs, we also suggest more advanced, special-purpose architecture courses in which the human and social aspects of meetings are explored side-by-side with the technical considerations that go into architecture design and evolution.

A critical ingredient of being a successful software architect is exposure to many different systems and their designs [7]. A specialized course in software architecture could begin to lay the foundations for students to engage in building a portfolio of systems and architectures to which they have been exposed. Accompanying such material with students actually making changes to the systems to which they are exposed, as exemplified by the work of van Deursen et al. [80], would further solidify their understanding of the role of architecture and how it shapes software.

VI. Threats to Validity

In this section, we discuss several threats to validity for our study.

Conclusion Validity. Threats to conclusion validity are concerned with issues that relate to the treatment and the outcomes of the study, including, for instance, the choice of sample size and, as another example, the care taken in the implementation of a study [81]. In our work, we conducted interviews with open-ended questions in which the participants were asked to provide their perceptions and point-of-views. The interviews were then corroborated through a survey. The interviews were conducted at 18 different companies and when they happened within the same company, the participants were warned not talk to each other about it to avoid bias. In addition, we requested and were given access to experienced software architects at each company, to avoid the interviewees not possessing the necessary deep and long-term experience and knowledge in our area of investigation. We approached the design and implementation of the survey with the same level of care.

Another aspect that is critical for conclusion validity is the quality of the material used in the study. Thus, to ensure that the interview prompt and survey instrument were of high quality, a pilot interview was conducted with a software architect and a survey pre-testing was performed with two software architects. Finally, to avoid the threat of drawing false conclusions based on the interview data, we carefully validated our interviews and findings with the participants as we performed analysis, asking for clarification when so needed.

Internal Validity. To reduce introducing interviewer bias during the interviews, we kept our questions open-ended and let participants talk most of the time. Additionally, it is possible that the participants might not have mentioned some points that, given more time to think, they could have brought up. To ameliorate this, we concluded the interviews by asking the participants whether they had any further thoughts and gave them ample time to respond before concluding the interviews. Similarly, we concluded the survey with a question as to whether survey participants had any additional thoughts they wanted to share.

Interviewing participants remotely might also introduce some bias as compared to interviewing in person, for instance, by the interviewees giving shorter, incomplete, or unclear answers. We attempted to reduce this bias by following up with the interviewees if we felt an answer needed more clarification for us to be able to understand it in retrospect.

Because they were derived from the answers from the interviews, a possibility exists that the questions on the survey might not have been sufficiently representative (e.g., additional aspects that change from initial whiteboard design as the architecture is further refined, additional approaches to document whiteboard software architecture meetings). This was mitigated by the option for the survey participants to provide additional thoughts through an open-ended question at the conclusion of the survey.

Finally, while our analysis was systematic, other researchers may discern aspects different from our findings.

Construct Validity. There are potential threats to construct validity from the lack of a clear definition of a software architect. In general, participants understood that we meant an experienced member in the development team responsible for making high-level design choices, validating them, and communicating those decisions to relevant stakeholders. In addition, we verbally clarified whenever there appeared to be some kind of confusion, both at the start of the interviews and throughout.

Another threat to construct validity is related to the potential problem of evaluation apprehension [81]. It was mitigated by letting the participants know they would remain anonymous as well as by assuring them that all information gathered during the interviews and survey would solely be used only by the research team and never shared beyond.

External Validity. Our 27 interviews were conducted with software architects working in 18 different companies. Though these interviews yielded important insights, it can be considered a small sample. In addition, we only sampled software architects from five countries and findings may not generalize to other countries and companies.

The same threat exists concerning the participants in the survey. Even though the respondents reside in nine countries across four continents, our findings may not generalize to represent the experiences and perceptions of all software architects.

VII. Conclusion

Becoming a software architect takes time and effort. In addition to having serious technical responsibilities in being the person who is primarily accountable for the software architecture and making architectural decisions, a software architect is also responsible for the many social aspects involved in the design and implementation of the architecture, which includes the subject of our study: conducting whiteboard software architecture meetings and bridging the outflow from these meetings to downstream activities. To date, such meetings have not been studied in detail and the realities of why these meetings are held, who typically participantes, how to best document the outcomes of the meetings, and what kind of downstream changes may occur and why are not fully understood yet.

In this paper, we contribute a mixed qualitative and quantitative study to investigate software architects' perceptions on whiteboard software architecture meetings. Based on interviews with 27 experienced architects and a subsequent survey with an additional 46 experienced software architects, our study yields eleven observations that range from reasons why software architects go to the whiteboard and perspectives on including experts and novices in the meetings, through how they document the outcomes of the meetings and why they document, to the kinds of changes they witness when the outcomes of the whiteboard meetings transition to downstream activities and the reasons for those changes. Our study is the first study of this kind, with the findings giving rise to further study, offering concrete advice for practicing architects, and suggesting new topics for educating future software architects.

As future work, we will engage in two activities. First, it is important to dive deeper into our results by contextualizing our findings with the content produced on the whiteboards during these meetings. The type of content produced can have implications for what kind of discussions are held, influence the strategies for capturing the discussions and their outcomes, and even be indicative of the quality of the discussion and thus the potential for problems downstream. Second, we wish to understand the role of novices in more detail and particularly study how they might move from being a novice in these meetings to a more full-fledged, equal participant. How does the nature of their contributions change over time, does their level of leadership change, and how do they engage in educating future novices?

ACKNOWLEDGMENT

The authors would like to thank all the software architects who participated in our interviews and survey.

REFERENCES

[1] M. Fowler, "Who needs an architect?," *IEEE Softw.*, vol. 20, no. 5, pp. 11–13, Sep. 2003.

[2] P. C. Clements, R. Kazman, M. Klein, D. Devesh, S. Reddy, and P. Verma, "The duties, skills, and knowledge of software architects," in *Proc. 6th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Mumbai, Maharashtra, India, Jan. 6–9, 2007, p. 20.

[3] M. E. Conway, "How do committees invent?," *Datamation*, Apr. 1968. Accessed: Sep. 23, 2023. [Online]. Available: http://www.melconway.com/research/committees.html

[4] C. Y. Baldwin and K. B. Clark, "Managing in an age of modularity," in *Managing in the Modular Age: Architectures, Networks, and Organizations*, vol. 149, Malden, MA, USA: Blackwell Publishers Ltd., 2003, pp. 84–93.

[5] F. P. Brooks Jr., "No silver bullet essence and accidents of software engineering," *Computer*, vol. 20, no. 4, pp. 10–19, Apr. 1987.

[6] D. E. Perry, N. A. Staudenmayer, and L. G. Votta, "People, organizations, and process improvement," *IEEE Softw.*, vol. 11, no. 4, pp. 36–45, Jul. 1994.

[7] P. Kruchten, "What do software architects really do?," *J. Syst. Softw.*, vol. 81, no. 12, pp. 2413–2416, 2008.

[8] J. F. Hoorn, R. Farenhorst, P. Lago, and H. van Vliet, "The lonesome architect," *J. Syst. Softw.*, vol. 84, no. 9, pp. 1424–1435, 2011.

[9] V. Clerc, P. Lago, and H. van Vliet, "The architect's mindset," in *Proc.3rd Int. Conf. Qual. Softw. Archit., QoSA: Softw. Archit., Compon., Appl.*, in Revised Selected Papers, Medford, MA, USA, Jul. 11–23, 2007, pp. 231–249.

[10] U. van Heesch and P. Avgeriou, "Mature architecting—A survey about the reasoning process of professional architects," in *Proc. 9th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Boulder, CO, USA, Jun. 20–24, 2011, pp. 260–269.

[11] K. Power and R. Wirfs-Brock, "Understanding architecture decisions in context—An industry case study of architects' decision-making context," in *Proc. 12th Eur. Conf. Softw. Archit. (ECSA)*, Madrid, Spain, Sep. 24–28, 2018, pp. 284–299.

[12] E. Woods, "Should architects code?" *IEEE Softw.*, vol. 34, no. 5, pp. 20–21, 2017.

[13] I. Rehman, M. Mirakhorli, M. Nagappan, A. A. Uulu, and M. Thornton, "Roles and impacts of hands-on software architects in five industrial case studies," in *Proc. 40th Int. Conf. Softw. Eng. (ICSE)*, Gothenburg, Sweden, May 27–Jun. 3, 2018, pp. 117–127.

[14] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. Hoboken, NJ, USA: Wiley, 2009.

[15] D. Falessi, M. A. Babar, G. Cantone, and P. Kruchten, "Applying empirical software engineering to software architecture: Challenges and lessons learned," *Empirical Softw. Eng.*, vol. 15, no. 3, pp. 250–276, 2010.

[16] P. O. Antonino, A. Morgenstern, and T. Kuhn, "Embedded-software architects: It's not only about the software," *IEEE Softw.*, vol. 33, no. 6, pp. 56–62, 2016.

[17] M. Erder and P. Pureur, "What's the architect's role in an agile, cloud-centric world?," *IEEE Softw.*, vol. 33, no. 5, pp. 30–33, Sep./Oct. 2016.

[18] E. Y.-L. Do and M. D. Gross, "Reasoning about cases with diagrams," in *Proc. 3rd Congr. Comput. Civil Eng.* Washington, DC, USA: ASCE, 1996, pp. 314–320.

[19] K. Henderson, *On Line and on Paper: Visual Representations, Visual Culture, and Computer Graphics in Design Engineering*. Cambridge, MA, USA: MIT Press, 1998.

[20] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *Proc. 28th Int. Conf. Softw. Eng. (ICSE)*. New York, NY, USA: ACM, 2006, pp. 492–501.

[21] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: How and why software developers use drawings," in *Proc. SIGCHI Conf. Human Factors Comput. Syst. (CHI)*. New York, NY, USA: ACM, 2007, pp. 557–566.

[22] N. Mangano, T. D. LaToza, M. Petre, and A. van der Hoek, "Supporting informal design with interactive whiteboards," in *Proc. SIGCHI Conf. Human Factors Comput. Syst. (CHI)*. New York, NY, USA: ACM, 2014, pp. 331–340.

[23] N. Mangano, T. D. LaToza, M. Petre, and A. van der Hoek, "How software designers interact with sketches at the whiteboard," *IEEE Trans. Softw. Eng.*, vol. 41, no. 2, pp. 135–156, Feb. 2015.

[24] T. Sharma, P. Singh, and D. Spinellis, "An empirical investigation on the relationship between design and architecture smells," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 4020–4068, 2020.

[25] R. Li, P. Liang, M. Soliman, and P. Avgeriou, "Understanding software architecture erosion: A systematic mapping study," *J. Softw. Evol. Process.*, vol. 34, no. 3, pp. 1–45, 2022.

[26] M. Petre and A. V. D. Hoek, *Software Designers in Action: A Human-Centric Look at Design Work*, 1st ed. London, U.K.: Chapman & Hall, 2013.

[27] U. Dekel, "Supporting distributed software design meetings: What can we learn from co-located meetings?," in *Proc. Workshop Human Social Factors Softw. Eng. (HSSE)*. New York, NY, USA: ACM, 2005, pp. 1–7.

[28] U. Dekel and J. D. Herbsleb, "Notation and representation in collaborative object-oriented design: An observational study," in *Proc. 22nd Annu. ACM SIGPLAN Conf. Object-Oriented Program., Syst., Lang., Appl. (OOPSLA)*, Montreal, Quebec, Canada. New York, NY, USA: ACM, Oct. 21–25, 2007, pp. 261–280.

[29] K. Nakakoji, Y. Yamamoto, N. Matsubara, and Y. Shirai, "Toward unweaving streams of thought for reflection in professional software design," *IEEE Softw.*, vol. 29, no. 1, pp. 34–38, Jan./Feb. 2012.

[30] J. Walny, J. Haber, M. Dörk, J. Sillito, and S. Carpendale, "Follow that sketch: Lifecycles of diagrams and sketches in software development," in *Proc. 6th Int. Workshop Visualizing Softw. Understanding Anal. (VISSOFT)*, 2011, pp. 1–8.

[31] S. Baltes and S. Diehl, "Sketches and diagrams in practice," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*. New York, NY, USA: ACM, 2014, pp. 530–541.

[32] S. Baltes, P. Schmitz, and S. Diehl, "Linking sketches and diagrams to source code artifacts," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*. New York, NY, USA: ACM, 2014, pp. 743–746.

[33] S. Baltes, F. Hollerich, and S. Diehl, "Round-trip sketches: Supporting the lifecycle of software development sketches from analog to digital and back," in *Proc. IEEE Work. Conf. Softw. Visualization (VISSOFT)*, 2017, pp. 94–98.

[34] D. Wüest, N. Seyff, and M. Glinz, "FLEXISKETCH TEAM: Collaborative sketching and notation creation on the fly," in *Proc. 37th IEEE/ACM Int. Conf. Softw. Eng. (ICSE)*, Florence, Italy, vol. 2. Los Alamitos, CA, USA: IEEE Comput. Soc., May 16–24, 2015, pp. 685–688.

[35] S. G. Samuelsson and M. Book, "Towards sketch-based user interaction with integrated software development environments," in *Proc. 42nd Int. Conf. Softw. Eng. (ICSE)* Workshops, Seoul, Republic of Korea. New York, NY, USA: ACM, Jun. 27–Jul. 19, 2020, pp. 181–184.

[36] F. Buschmann, "A week in the life of an architect," *IEEE Softw.*, vol. 29, no. 3, pp. 94–96, May/Jun. 2012.

[37] E. Woods, "Return of the pragmatic architect," *IEEE Softw.*, vol. 31, no. 3, pp. 10–13, May/Jun. 2014.

[38] J. Klein, "What makes an architect successful?," *IEEE Softw.*, vol. 33, no. 1, pp. 20–22, Jan./Feb. 2016.

[39] M. Erder and P. Pureur, "What type of people are software architects?," *IEEE Softw.*, vol. 34, no. 4, pp. 20–22, 2017.

[40] J. Klein, "How does the architect's role change as the software ages?," in *Proc. 5th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Pittsburgh, PA, USA, Nov. 6–10, 2005, p. 141.

[41] P. Sarang, "Setting up architect team," in *Proc. 6th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Mumbai, Maharashtra, India, Jan. 6–9, 2007, p. 18.

[42] R. Premraj, G. Nauta, A. Tang, and H. van Vliet, "The boomeranged software architect," in *Proc. 9th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Boulder, CO, USA, Jun. 20–24, 2011, pp. 73–82.

[43] I. Seidman, *Interviewing as Qualitative Research: A Guide for Researchers in Education and the Social Sciences*, 3rd ed., New York, NY, USA: Teachers College Press, 2006.

[44] J. Saldana, *The Coding Manual for Qualitative Researchers*. Newbury Park, CA, USA: Sage, 2015.

[45] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[46] J. Mason, *Qualitative Researching*. Newbury Park, CA, USA: Sage, 2018.

[47] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to Advanced Empirical Software Engineering*, New York, NY, USA: Springer, 2008, pp. 63–92.

[48] P. Clements et al., *Documenting Software Architectures: Views and Beyond*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2010.

[49] J. van Gurp and J. Bosch, "Design erosion: Problems and causes," *J. Syst. Softw.*, vol. 61, no. 2, pp. 105–119, 2002.

[50] N. Ali, S. Baker, R. O'Crowley, S. Herold, and J. Buckley, "Architecture consistency: State of the practice, challenges and requirements," *Empirical Softw. Eng.*, vol. 23, no. 1, pp. 224–258, 2018.

[51] J. H. Lee and M. J. Ostwald, "The relationship between divergent thinking and ideation in the conceptual design process," *Des. Stud.*, vol. 79, 2022, Art. no. 101089.

[52] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, pp. 40–52, Oct. 1992.

[53] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[54] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Reading, MA, USA: Addison-Wesley, 2003.

[55] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, "Documenting software architectures: Views and beyond," in *Proc. 25th Int. Conf. Softw. Eng.* Piscataway, NJ, USA: IEEE, 2003, pp. 740–741.

[56] P. Clements et al., *Evaluating Software Architectures*. Beijing, China: Tsinghua University Press, 2003.

[57] A. Tang, P. Liang, and H. v. Vliet, "Software architecture documentation: The road ahead," in *Proc. 9th Work. IEEE/IFIP Conf. Softw. Archit.*, 2011, pp. 252–255.

[58] B. Reimlinger, Q. Lohmeyer, R. Moryson, and M. Meboldt, "A comparison of how novice and experienced design engineers benefit from design guidelines," *Des. Stud.*, vol. 63, pp. 204–223, 2019.

[59] X. Ge, L. Leifer, and L. Shui, "Situated emotion and its constructive role in collaborative design: A mixed-method study of experienced designers," *Des. Stud.*, vol. 75, 2021, Art. no. 101020.

[60] E. M. Silk, A. E. Rechkemmer, S. R. Daly, K. W. Jablokow, and S. McKilligan, "Problem framing and cognitive style: Impacts on design ideation perceptions," *Des. Stud.*, vol. 74, 2021, Art. no. 101015.

[61] S. Krishnakumar, C. Berdanier, C. Lauff, C. McComb, and J. Menold, "The story novice designers tell: How rhetorical structures and prototyping shape communication with external audiences," *Des. Stud.*, vol. 82, 2022, Art. no. 101133.

[62] C. A. Gorse and S. Emmitt, "Communication behaviour during management and design team meetings: A comparison of group interaction," *Constr. Manage. Econ.*, vol. 25, no. 11, pp. 1197–1213, 2007.

[63] S. B. Paletz, J. Chan, and C. D. Schunn, "The dynamics of micro-conflicts and uncertainty in successful and unsuccessful design teams," *Des. Stud.*, vol. 50, pp. 39–69, 2017.

[64] A. Tang, M. A. Babar, I. Gorton, and J. Han, "A survey of architecture design rationale," *J. Syst. Softw.*, vol. 79, no. 12, pp. 1792–1804, 2006.

[65] O. Kopp, A. Armbruster, and O. Zimmermann, "Markdown architectural decision records: Format and tool support," in *Proc. 10th Central Eur. (CEUR) Workshop Services Their Compos.*, Dresden, Germany, vol. 2072. CEUR-WS.org, Feb. 8–9, 2018, pp. 55–62.

[66] S. D'Angelo and D. Gergle, "An eye for design: Gaze visualizations for remote collaborative work," in *Proc. CHI Conf. Human Factors Comput. Syst.*, Montreal, QC, Canada. New York, NY, USA: ACM, Apr. 21–26, 2018, p. 349.

[67] D. Ford et al., "A tale of two cities: Software developers working from home during the COVID-19 pandemic," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 2, pp. 27:1–27:37, 2022.

[68] A. M. Soria, A. van der Hoek, and J. E. Burge, "Recurring distributed software maintenance meetings: Toward an initial understanding," in *Proc. 15th IEEE/ACM Int. Workshop Cooperative Human Aspects Softw. Eng., CHASE@ICSE*, Pittsburgh, PA, USA. Piscataway, NJ, USA: IEEE, May 21–22, 2022, pp. 21–25.

[69] A. M. Soria, "KNOCAP: Capturing and delivering important design bits in whiteboard design meetings," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.: Companion Proc. (ICSE)*. New York, NY, USA: ACM, 2020, pp. 194–197.

[70] M. Petre and A. Van Der Hoek, *Software Design Decoded: 66 Ways Experts Think*. Cambridge, MA, USA: MIT Press, 2016.

[71] L. Delizonna, "High-performing teams need psychological safety. Here's how to create it," *Harvard Bus. Rev.*, vol. 8, pp. 1–5, 2017.

[72] A. Newman, R. Donohue, and N. Eva, "Psychological safety: A systematic review of the literature," *Human Resou. Manage. Rev.*, vol. 27, no. 3, pp. 521–535, 2017.

[73] P. Lago and H. van Vliet, "Teaching a course on software architecture," in *Proc. 18th Conf. Softw. Eng. Educ. Training (CSEET)*, 2005, pp. 35–42.

[74] A. Van Deursen et al., "A collaborative approach to teaching software architecture," in *Proc. ACM SIGCSE Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2017, pp. 591–596.

[75] J. I. Benedetto and J. Navón, "Exploiting group shuffling dynamics to convey the importance of good software design," in *Proc. 42nd Int. Conf. Softw. Eng., Softw. Eng. Educ. Training (ICSE-SEET)*, Seoul, South Korea. New York, NY, USA: ACM, Jun. 27–Jul. 19, 2020, pp. 193–196.

[76] Z. Li, "Using public and free platform-as-a-service (PaaS) based lightweight projects for software architecture education," in *Proc. 42nd Int. Conf. Softw. Eng., Softw. Eng. Educ. Training (ICSE-SEET)*, Seoul, South Korea. New York, NY, USA: ACM, Jun. 27–Jul. 19, 2020, pp. 1–11.

[77] S. A. Rukmono and M. R. V. Chaudron, "Guiding peer-feedback in learning software design using UML," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng.: Softw. Eng. Educ. Training (ICSE-SEET)*, Pittsburgh, PA, USA. Piscataway, NJ, USA: IEEE, May 22–24, 2022, pp. 122–133.

[78] M. Petrenko, D. Poshyvanyk, V. Rajlich, and J. Buchta, "Teaching software evolution in open source," *Computer*, vol. 40, no. 11, pp. 25–31, 2007.

[79] K. Gallagher, M. Fioravanti, and S. Kozaitis, "Teaching software maintenance," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, 2019, pp. 353–362.

[80] A. van Deursen et al., "A collaborative approach to teaching software architecture," in *Proc. ACM SIGCSE Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, M. E. Caspersen, S. H. Edwards, T. Barnes, and D. D. Garcia, Eds., Seattle, WA, USA. New York, NY, USA: ACM, Mar. 8–11, 2017, pp. 591–596.

[81] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. New York, NY, USA: Springer, 2012.

Eduardo Santana de Almeida (Senior Member, IEEE) received the Ph.D. degree in computer science from the Federal University of Pernambuco, Brazil. He is currently an Associate Professor with the Institute of Computing (IC), Federal University of Bahia, where he leads the RiSE Labs. His research interests include software reuse, software product lines, software architecture, and empirical software engineering. He is a senior member of ACM and an affiliate member of the Brazilian Academy of Sciences (ABC).

Iftekhar Ahmed received the B.Sc. degree in computer science and engineering from Shahjalal University of Science and Technology, Bangladesh, and after working in the industry for four years, received the Ph.D. degree from Oregon State University. He is currently an Assistant Professor in informatics with the Donald Bren School of Information and Computer Science, University of California, Irvine. His research interests include the intersection of software engineering and machine learning, encompassing socio-technical factors, and large-scale analysis of software artifacts to ensure software quality.

André van der Hoek (Member, IEEE) received the B.S. and M.S. degrees in business-oriented computer science from Erasmus University Rotterdam, The Netherlands, and the Ph.D. degree in computer science from the University of Colorado at Boulder. He is a Professor with the Department of Informatics at the University of California, Irvine, and the Head of the Software Design and Collaboration Laboratory, which focuses on understanding and advancing the roles of design, collaboration, and education in software engineering. He is a Co-Author of *Software Design Decoded: 66 Ways How Experts Think* and an Co-Editor of *Studying Professional Software Design: A Human-Centric Look at Design Work*, two books that detail the expert practices of professional software designers. He has authored and co-authored over 100 peer-reviewed journal and conference publications. In 2006, he was a recipient of an ACM SIGSOFT Distinguished Paper Award; in 2013, he was recognized as an ACM Distinguished Scientist; and in 2009, he was a recipient of the Premier Award for Excellence in Engineering Education Courseware. He is the Principal Designer of the B.S. in informatics at UC Irvine. He was honored, in 2005, as UC Irvine Professor of the Year for his outstanding and innovative educational contributions.