We're Drifting Apart: Architectural Drift from the Developers' Perspective

Emilie Anthony, Astrid Berntsson
Chalmers | University of Gothenburg
Gothenburg, Sweden
{emilie.anthony1, astridberntsson1}@gmail.com

Tiziano Santilli
Gran Sasso Science Institute (GSSI)
L'Aquila, Italy
tiziano.santilli@gssi.it

Rebekka Wohlrab
Chalmers | University of Gothenburg
Gothenburg, Sweden
wohlrab@chalmers.se

***Abstract*— Despite the recognized importance of software architecture, it is common that the implementation diverges from the intended architecture over time. This phenomenon is referred to as architectural drift. In the past decades, mainly technical solutions and tools have been developed to detect and address architectural inconsistencies and drift. There is still a lack of evidence from the perspective of developers and a lack of best practices to manage drift. This mixed-methods study relies on interviews with 11 developers and a survey answered by 63 developers from different companies and domains. We analyzed the data by dividing developers into senior and junior to see the different perspectives based on work experience. We found that juniors tend to rely more on documentation, while seniors have a more experience-related approach. We identified practices that developers use to mitigate drift, including defining clear responsibilities, setting best practices, and maintaining reliable documentation. Finally, we designed and evaluated guidelines to help developers to face architectural drift.**

***Index Terms*—Architectural drift, survey, interviews**

I. Introduction

The importance of software architecture in software engineering is widely recognized [1]. The most essential characteristics and design limitations of software systems are often captured in software architectures [2]. Ensuring that the envisioned architecture is consistently implemented is challenging, despite thorough documentation [3]. As software systems are developed and maintained, and new requirements emerge, architectural drift is commonly introduced. *Architectural drift* occurs when a software system evolves and gradually moves from an intended architecture to a different unintended architecture [4]. While architectural erosion is caused by direct violations of the architecture, drift occurs due to modifications that are not violations, but still introduce inconsistencies with the architecture [4], [5]. Based on the literature, we have provided a visual summary of architectural drift in Fig. 1. It can be seen that an initially *implemented architecture* is consistent with the *intended architecture*. Then, due to various *causes*, *inconsistencies* with the initially intended architecture are introduced while creating a *new implemented drifted architecture*. In this process, drift is caused which has several *possible consequences*.

While there exist few studies focusing explicitly on architectural drift, architectural degeneration and deviation have received considerable attention [6], [5] with proposals of various tools to maintain consistency between architecture and implementation [7], [8], [9]. Focusing on architectural consistency, there have been multiple studies conducted with a focus on the perspective of the software architect [3], [10]. Recently, there has been an increased interest in investigating architectural deviations from the developers' perspective [11]. As the people who carry out in-situ decisions when implementing and changing code, developers have a significant impact on architectural drift [12]. Often being part of empowered agile teams, developers play a central role in the development of software systems, and is important to investigate architectural drift from their perspective. They are also often a part of development teams and interact with architects and other roles [13]. In this paper, we aim to further understand the developers' perspective on architectural drift, considering both technical and non-technical factors. Non-technical factors have considerable importance in the manifestation of deviations [11]. However, technical research on architectural degradation is the dominant part of published articles [6]. Considering that drift is a complex and multifaceted phenomenon from the developers' point of view, additional empirical research can contribute to a better understanding of its characteristics.

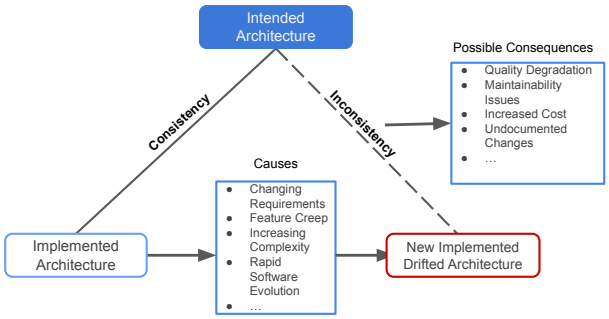

Fig. 1. Visual summary of architectural drift [4]

To investigate how developers reason about and handle architectural drift, we conducted a mixed-methods study with 11 semi-structured interviewees [14]. Based on the findings from the interviews, we created a survey and received 63 responses [15]. This work expands on the developers' perspective and further identifies both technical and non-technical areas. To understand how developers with different levels of

experience can be better supported, we distinguish between senior and junior developers in our analysis of architectural drift. We address the following research questions:

- **RQ1: What are senior and junior developers' challenges regarding architectural drift?**
- **RQ2: What practices do senior and junior developers currently use to mitigate architectural drift?**
- **RQ3: What practices do senior and junior developers regard as beneficial to mitigate architectural drift?**

The study reveals that developers often rely on established ways of working to detect, address, and prevent architectural drift and see an advantage in processes and conventions. We found that junior developers tend to rely more on documentation, while seniors have a more experience-related approach. Besides defining clear responsibilities, setting best practices, and maintaining reliable documentation, it was stressed that it would be beneficial to document the reasons behind decisions. At the same time, currently, the documentation of rationales is often lacking in our participants' companies.

II. Background and Related Work

The term architectural drift is often used to refer to the phenomenon of when the implemented architecture deviates from the intended architecture, in a way that is caused by insensitivity to the architecture [4]. The intended architecture refers to the planned, prescriptive, or as-designed architecture. Usually, it is conceived by software architects and documented, for example, using UML diagrams. On the other hand, the implemented architecture represents the as-implemented, as-built, descriptive, or as-realized architecture [16]. Commonly, developers make ad-hoc decisions when implementing a software system, and their decisions impact how different components are connected and what architectural decisions can be found in the implemented system. Several terms have been used to describe architectural deviation in the literature, e.g., drift, erosion, degeneration, mismatch, degradation, design decay, and inconsistency [17], [12], [2], [3], [6], [18]. Wolf and Perry [4] provided two interconnected definitions: *drift*, stemming from neglect or insensitivity to the architecture, and *erosion*, resulting from violations of the architecture.

Architectural drift can hinder the evolution of systems. Even if recognized, addressing this drift can entail a costly resolution process [3]. Quality degradation, maintainability issues, increased cost, and undocumented changes are some consequences of drift mentioned in literature [3], [19]. Due to the negative impacts of architectural drift, several methods and tools have been designed. Connected to both erosion and drift, Whiting and Andrews identified three strategies to solve them: organizational competence, proactive methods, and reactionary methods [19]. Organizational competence is connected to enabling teams to design high-quality architectures. Its focus is not only to educate architects but also to make developers organizationally competent and enable them to perform good design practice. The strategy of organizational competence is aligned with the focus of this paper: we investigate the developers' perspective, since they are roles that might be consciously or subconsciously involved with architectural drift. The second strategy, proactive methods, is related to technical solutions that help stakeholders design better architectures and counteract drift in the first place. Those solutions might rely on NLP techniques or the selection of particular architectural styles. Reactionary techniques, on the other hand, are supposed to minimize drift when it occurs. Code comments, static analysis tools, or conformance checking are examples of reactionary techniques.

Whiting and Andrews's classification of strategies was mainly based on a review of related work. In this paper, we aim to complement their analysis by presenting empirical evidence from real companies and developers.

TABLE I
Interview participants (all of which are developers).

No	Developer Role	Exp. (yrs)	Domain	Company	Size
P1	Fullstack	5	E-commerce	A	Large
P2	Fullstack	5	Logistics	B	Medium
P3	DevOps	2	Automotive	C	Large
P4	Embedded	12	Telecom	D	Medium
P5	Fullstack	8	Industrial	E	Micro
P6	Backend	<1	Automotive	C	Large
P7	Backend	20	E-commerce	F	Small
P8	Fullstack	<1	IoT	G	Medium
P9	Backend	2	Business management	H	Large
P10	Frontend	<1	Automotive	I	Large
P11	Backend	5	Automotive	I	Large

III. Research Method

Given that drift is a complex phenomenon, we opted for a mixed-methods study with exploratory research questions. We used a mixed-methods study to get rich qualitative data from semi-structured interviews and triangulate the data with responses from a survey. The steps of our method are shown in Fig. 2. In total, we performed 11 interviews with developers and a survey that was answered by 63 developers.

Fig. 2. Overview of the research methodology

A. Selection of Interview Participants

In our selection of interview participants, we made use of purposive sampling [20]. Our selection criteria focused on interviewees having a minimum of six months of professional experience as developers. We set the limit to six months to also include junior developers with some work experience. We made a concerted effort to include interviewees from different backgrounds, encompassing various company sizes, domains, and experience levels. Table I shows an overview of the 11 selected interviewees. The interviewees' companies are active within seven different domains including automotive, logistics, e-commerce, and telecommunication. The participants' work experience ranges from six months to 20 years, with an average of 5.5 years. Among the selected participants, 45%

identify as women, and 55% identify as men. As for the various company sizes, 6 out of 11 developers work in a large company, 3 out of 11 work in a medium-sized company, and 2 out of 11 work in a small or micro-sized company, according to the European classification standard[1].

B. Selection of Survey Participants

For the participant selection, we broadly searched for developers with at least six months of experience in a software company. We selected the participants using direct contacts from our professional network and via Facebook groups that focus on software developers. For the direct contacts, most of the participants came from Sweden and Italy. For the Facebook groups, we shared the survey in groups of mixed nationalities, and in specific groups by language, nationality, and programming languages, to have a wider sample.

The software developers who responded to the survey have an average work experience of 7.17 years. We represented multiple genders, 18% of the participants identify as women, 79% identify as men, and 3% identify as non-binary. As for the nationality of the participants, 37 are Swedish, 17 are Italian, four are Spanish, one is Canadian, one is Swiss, one is from UAE, one is from the UK, and one is from the Netherlands. Regarding the size of the company where the developers are working, 46% work in large companies, 24% in medium-sized companies, 24% in small companies, and 6% in micro companies, according to the European classification standard[1]. The domains in which developers work the most were automotive, as well as web and mobile applications.

C. Interviews

Before conducting the interviews, we constructed an interview guide [14] where we categorized our questions according to the research questions. At the start of the interview, we asked participants what architectural drift is and provided them with the same definition used in this paper to establish common ground: *"Architectural drift is the phenomenon in which a software system evolves, gradually moving from an initially planned architecture to a different unintended architecture."* We stressed that drift is not caused by direct violations. We improved the interview guide by incorporating feedback from two pilot interviews. Those adjustments included clarification of questions and adding Likert-scale [21] statements to add complementary data and spark discussions. The interview guide is included in the supplementary material[2].

When we reached out to the participants, we explained the purpose of the interview and how the data would be handled. Three participants were provided with the interview guide before the interviews as this was requested by them. The interviews were conducted online via video call and took 40 minutes on average. We recorded the audio of all of the interviews and manually transcribed them. Before starting the recording, the participants were assured anonymity and confidentiality.

[1]https://single-market-economy.ec.europa.eu/smes/sme-definition

[2]doi.org/10.6084/m9.figshare.23808087

D. Survey

We created a 15-minute survey using SoSciSurvey[3], ensuring anonymity for all participants. To reduce biases and increase the validity of the survey, we went through various iterations. First, we carefully discussed and refined the questions internally within the research team. In a second iteration, we consulted two external software developers to refine the survey [15].

The survey contains exclusively questions and sentences that we have extrapolated from the interviews, and is used to validate the data we have collected.

The survey consists of 5 pages. On the first page, we ask if the respondent writes code in their work and clarify what we mean by the term drift. We provided the same definition as the one used in the interviews. Page 2 includes a series of questions on the challenges that we found in the interviews. Page 3 contains questions regarding the current practices that are used by developers for detecting, handling, and preventing drift, as well as their opinion on who is, or should be responsible for managing drift. Page 4 of the survey allows the developer to rate several guidelines that could help in the management and prevention of drift. The last page of the survey contains demographic questions. For the creation of the questions, we used the United Nations Guidelines for gender-inclusive language [22]. A complete copy of the survey is included in the supplementary material[2].

E. Qualitative Analysis

For this exploratory study, we relied on qualitative data analysis methods [23]. We used thematic analysis according to Braun and Clark's [24], [25] practical guide and followed the six phases: familiarization with the data (1), coding (2), initial theme development (3), developing, reviewing, and refining themes (4), defining and naming themes (5), and writing (6). Since the research questions in this study are exploratory, we used mostly an inductive approach to analyze the data. To verify and refine the analysis, we used a deductive approach using a priori codes to iterate over the material. The a priori codes were based on the interview guide and the research questions. We used semantic coding with a focus on the explicit content of the data and did not look for any underlying meanings.

Thematic analysis is a flexible method that can be beneficial when analyzing different types of interview data, such as semi-structured or unstructured interviews. It allowed us to identify the most important themes and sub-themes that we generated from the data. Thematic analysis also offers transparency as it allowed us to document the analytical process, making it easier to evaluate the findings.

1) Inductive phase: We started by familiarizing ourselves with the data (1). Once the 11 interviews had been recorded, two of the authors manually transcribed them into text. Generating initial codes created of a piece of data that was relevant to answering the research questions (2). The coding

[3]https://www.soscisurvey.de

was directed by the content of the data. The two first authors initially coded the data individually while continuously making notes to decrease bias. Afterwards, we coded using the collaborative features of Atlas.ti[4]. We compared the codes from the individual coding phase and discussed their meaning for consistency. We compared the individual results and continued to search for themes by merging the codes and grouping them. We continued with searching for and constructing themes by identifying codes that share common meanings and clustering those codes together around a central idea (3). To do so we used the collaborative platform Miro[5]. We reviewed the themes by ensuring they answered the research questions (4). Based on our research questions and the interview guide, we used our identified themes to search for new codes, and define and generate descriptive names for the themes (5).

2) Deductive phase: We complemented the six-phase method with a deductive approach by coding and developing themes that were directed by existing codes and themes. This was done to capture excerpts that fit the already existing codes. Although we ran through earlier steps (1-3), we focused mainly on refining (4), defining and naming (5), and writing (6) during this phase. In the last step, we finalized the writing of this paper and selected quotes that supported our themes (6). The resulting themes, theme descriptions, codes, and code descriptions can be found in the supplementary material.

3) Example: To illustrate the coding process of the data, we provide this quote as an example from P1 (Participant 1):

> *"We're [...] trying to get documentation down because we took over a project from a previous team and they hadn't documented much. We're currently trying to get that documentation down and figure out 'what is the architecture?' [...] and see what changes we can make without breaking anything."* (P1)

This was initially coded as Unreliable Documentation, Inherit Project, and Handling Unclear Intent. The interviewee mentioned that they took over a project and "they hadn't documented much" and therefore assigned the code Unreliable Documentation and Inherit Project. The interviewee also mentions "Why is it here?" which we interpreted as not knowing why certain architectural decisions were made, thus assigning Handling Unclear Intent. These codes became part of the two themes Navigation in Uncertainty and Incomplete Documentation which were mapped to the first research question. Our codebook is available in our supplementary material[2].

IV. Findings

We report the findings of our research questions below. Due to limited space, we will only describe the underlined themes that we found to be most relevant. The themes created during the thematic analysis are presented in Tables II, IV, and VII with their corresponding survey question. Demotivation and onboarding are general issues in software engineering [26], [27] and are not directly connected to drift, so we decided not to investigate these issues further in the survey. For each category of questions, we divided the answers into totals (complete number of participants), Junior (Experience $<$ 5 years), and Senior (Experience $\geq$ 5 years). We also added the difference in percentage points (pp) between junior and senior developers. Of the 63 developers who responded to the survey, 38 are in the senior category, and 25 are in the junior category. All analysis results can be found in the supplementary material[2].

A. Challenges (RQ1)

Our interview data revealed various challenges which are presented in Tab. II. The survey results related to developers' perceived challenges are shown in Tab. III.

Apart from the data shown in those tables, we also asked the survey respondents what they considered to be the biggest challenges. 63% answered that prioritizing architecture work was the most challenging issue.*"Incomplete or missing documentation"* was selected by 47%.

Seniors and juniors have different views regarding the most relevant challenges. Senior developers consider prioritization and the lack of architectural knowledge as most critical. Junior developers consider documentation as more crucial.

In the following, we describe selected challenges in further detail.

1) Prioritization: When considering all of the different constraints and objectives of an organization, it often ends up being a challenge for the developers to prioritize architectural work. Business and engineering objectives do not always align, creating a misalignment of expectations and resulting in developers having to prioritize one over the other. P7 (Participant 7) describes how prioritizing often results in engineering practices, such as documentation, being neglected:

> *"[We ask ourselves] 'should we document or deliver?' and the focus is always on delivery. Then you abandon the documentation. That almost always happens."* (P7)

The fact that 63% of the survey respondents consider prioritization the most challenging issue indicates that companies need to establish aligned priorities. Prioritizing architectural consistency is important to many respondents. 68% of the participants in the survey agreed or strongly agreed with the statement that *"keeping the code aligned with the intended architecture is a priority for my team"*.

2) Allocated resources: The developers' work is directly impacted by business constraints, objectives, and allocated resources, leading to challenges in mitigating architectural drift. P11 explained: *"It never depends on the software developer. It always depends on the budget and what the business has in consideration"*. This is echoed by other interviewees and P9 explained that sometimes documentation is not prioritized by customers, partially because of the budget: *"Some customers don't value documentation and then we don't write documentation."*

To save time by doing things quickly and delivering within a certain time frame, developers sometimes deviate from

[4]https://atlasti.com
[5]https://miro.com

TABLE II
CHALLENGES NAMED IN THE INTERVIEWS WITH ASSOCIATED SURVEY QUESTION(S). UNDERLINED PRACTICES ARE EXPLAINED IN DETAIL.

Theme	Description	Representative Quote	Survey Question
Prioritization	Relates to challenges related to prioritization between business and engineering objectives in the developer team.	"The architecture can change over time [...] due to pressure from the business side. It's the eternal struggle between product and engineering: we have to build features now, and we want to release them quickly and [not compromise the quality of the code]." (P2)	2
Allocated resources	Refers to challenges related to resources allocated to developers (e.g., lack of time and budget constraints).	"There I chose to depart from the architecture [...] because it started to become deadline critical." (P9)	3
Inexperience	Challenges related to mitigating drift due to lack of working experience or being newly hired.	"If you are new to an undocumented project like I am, the structure is more of a mystery than anything else." (P1)	5
Demotivation	The motivation of developers and its effect on the architecture.	"You want motivated developers who enjoy working. If you don't have that, the quality won't be so good either." (P7)	N/A
Incomplete documentation	Challenges related to documentation being incomplete (e.g., it is outdated, lacks information, or does not accurately reflect the current implementation).	"There is a lot of documentation, but it is sometimes several years old and completely out of date. That's been one of the big problems since I started, that nothing [in the code] matches the documentation." (P7)	9 & 4
Navigation in uncertainty	Relates to challenges developers experience when making decisions based on assumptions (e.g., because of lack of information or knowledge).	"You learn things along the way and the implementation just doesn't turn out the way you intended. [...] you've added a few features and realize that it doesn't scale at all and we can't read what we're doing anymore." (P2)	6
Flawed communication	Refers to challenges related to written and verbal communication within and between teams in relation to drift.	"The communication can be very challenging. [...] Drift is much easier to discuss in a mature team where everyone is comfortable with each other." (P2)	7
Issues in implemented code	Challenges that are related to the code implementation, e.g., difficulties extending, testing, and reading the code.	"We had to add a new feature, but could not extend the code. We had to modify some of the code changes because the code was becoming too complex to test and maintain." (P11)	10
Difficulties onboarding	Refers to challenges onboarding new team members.	"Onboarding new members can be tricky if there is a drift in the architecture. We cannot rely on the documentation so [new developers] can't get up to date with the current state of the art of the system." (P4)	N/A

the intended architecture. P9 explained:*"There I chose to depart from the architecture [...] because it started to become deadline critical"*. On a question on how to manage drift, P8 mentions restricted time as a factor: *"It depends on deadlines and the people responsible"*.

59% of survey participants stated that they deviate from the intended architecture due to a lack of time. Senior developers agree more (47%) with the statement compared to junior developers (32%), which indicates that senior developers are more aware of the effort-benefit tradeoff that comes with counteracting drift.

3) Incomplete documentation: Several interviewee participants mentioned the challenge of documentation being incomplete. P1 described inheriting a code base and highlighted the difficulties of working with a system without comprehensive knowledge of its underlying design principles:

> *"We don't know why the architecture looks the way it does and we don't always know how it works so much of what we do is to try to be very careful to avoid breaking it."* (P1)

Several developers mentioned inheriting projects, from another group of developers, without sufficient documentation as being especially challenging. P7 stated:

> *"They never documented anything either. [...] It was a big problem that the documentation was extremely poor."* (P7)

P2 talked about the challenge of losing relevant knowledge:

> *"If a person leaves, that information disappears. [...] I specifically think about information of which other systems our system communicates with, what architectural decisions have we made, and why we made them."* (P2)

When asked what they see as the biggest challenge with drift, 60% of the junior developers selected "incomplete or missing documentation". Among the senior developers, it was selected as the top challenge by 39%. According to our findings, senior developers generally consider communication as more relevant than documentation. This insight is also reflected in the data shown in Tab. III. 88% of the developers stated that *"having bad quality documentation increases the likelihood that the implementation deviates from the intended architecture"*. Compared to senior developers, junior developers consider outdated or incomplete documentation more problematic.

TABLE III
SURVEY RESPONDENTS AGREEING WITH STATEMENTS ON CHALLENGES.

Statement	Total	Junior	Senior	Difference
1. "My team shares a common understanding of the intended architecture."	77%	80%	76%	4pp
2. "Keeping the code aligned with the intended architecture is a priority for my team."	68%	64%	71%	-7pp
3. "Sometimes, I don't follow the intended architecture because it would take me too much time."	41%	32%	47%	-15pp
4. "Having bad quality documentation increases the likelihood that the implementation deviates from the intended architecture."	85%	88%	84%	4pp
5. "I believe that more experienced developers mitigate architectural drift better than less experienced developers."	74%	78%	71%	7pp
6. "Sometimes, when writing code, I improvise and make assumptions on what the architecture should look like."	69%	60%	75%	-15pp
7. "Sometimes, it's a problem for us that we don't talk about architectural decisions."	89%	83%	92%	-9pp
8. "I think it is a problem if the code deviates from the intended architecture."	73%	76%	71%	5pp
9. "Having outdated documentation is a big problem for us."	64%	72%	58%	14pp
10. "Architectural drift creates quality issues."	66%	68%	65%	3pp

4) Navigation in uncertainty: During the initial (re-)design of an architecture, there is a high degree of uncertainty and a lot of assumptions have to be made. Developers also make assumptions in situations where they do not have accurate architectural information available and lack an understanding of the intentions behind architectural choices. Our participants described that they commonly make architectural decisions and implement the architecture based on assumptions. P2

explained this issue as follows:

> *"You might have an idea that 'yes, but this pattern will work really well', then a few months go by and you've added a few features and realize that it doesn't scale at all and we can't read what we're doing anymore."* (P2)

P6 mentioned that new insights could change the requirements and that leads to *"[...] a quite big drift from the initial draft of plans"*. More senior developers report that they improvise (75%) compared to junior developers (60%).

5) Issues in implemented code: Issues in implemented code encompass various challenges that the developers mentioned they encounter in their work. These issues include bugs, increased complexity, increased difficulty in testing, and deteriorated readability. P2 described the consequences of a drifted implementation based on assumptions made early on:

> *"We sprinkled customer-specific logic everywhere which made the code very difficult to work with in the end. The bugs that originated from that system became incredibly difficult to fix."* (P2)

Furthermore, P9 stated: *"I would say that the challenges lie in the difficulty of code maintenance and troubleshooting"*.

6) Flawed communication: Several interviewees brought up the difficulty in communication. Miscommunication can lead to information not being shared. P1 stated: *"I often find that communication is the problem in many situations—it fails."* Developing in isolation without input from others was raised by interviewee P9: *"But in my company, we work too loosely and independently with architecture."* P2 stated:

> *"The communication can be very challenging. [...] Drift is much easier to discuss in a mature team where everyone is comfortable with each other."* (P2)

89% of the survey respondents indicated that it sometimes is a problem for them that they don't talk about architectural decisions. More senior developers (92%) think that this is a problem compared to junior developers (83%).

Summary (RQ1): We conclude that the lack of documentation is a prevalent challenge, followed by other issues such as communication. Junior developers perceive documentation as more important than seniors, who consider communication as critical and are more willing to improvise architectural decisions to save time.

B. Current Practices (RQ2)

Our findings on current practices from the interviews are presented in Tab. IV. Tab. V and VI show the survey responses regarding the currently used practices. It can be seen that there are differences between junior and senior developers. More senior developers (62%) than junior developers (42%) agree with the statement that *"architectural Drift is acceptable"*. Junior developers, on the other hand, more commonly agree with the statement that they *"have sufficient practices in place to handle architectural drift"* (75%) in comparison to senior developers (57%). These findings indicate that the level of experience changes developers' perceptions of drift.

According to the survey, the most used practice to detect and prevent drift is *"reviewing other people's code"* and *"sharing information with others"*. Senior developers tend more to work alone with the code to detect and prevent drift than junior developers (42% and 32%, respectively). Overall, 11% of the respondents report that they don't have a dedicated practice. Junior developers more commonly state that they do not have any dedicated practices than senior developers (16% and 8%, respectively). In the interviews, none of the participants mentioned that they use dedicated tools to counteract architectural drift.

1) Practices to detect and prevent drift: Developers consider themselves to be aware of when drift occurs. As seen in Tab. VI, when asked to agree or disagree with the statement *"I am aware when the code drifts from its intended architecture"*, 87% of the respondents agreed or strongly agreed. Interviewee participant P4 commented on the matter: *"[...] I would say that I am not aware at that moment, I eventually become aware at some other time in the future."* Different practices are used to detect and prevent drift:

a) Code reviews: Multiple participants mentioned code reviews as a way to both detect and prevent architectural drift. Code reviews were especially emphasized by the interviewees as a key practice. P2 explained how drift is detected: *"I believe that it could be either because I have implemented something myself or when I'm reviewing a pull request."*. Another example is:

> *"I rely on the manual eye and place a lot of emphasis on code reviews because that's where a lot [of deviations] can be caught."* (P1)

73% of the survey participants state that they review other people's code to detect and prevent drift.

b) Implementing code: Developers described difficulty implementing and extending code as a way to become aware of drift:

> *"As a developer, it's when I feel that I have to forcefully incorporate functionality into my code. [...] Then it's something that the architecture no longer supports."* (P2)

38% of the survey respondents report that they detect or prevent drift when working with the code. More senior developers work alone with the code to detect and prevent drift than junior developers.

c) Discussions: Discussions have a role in preventing drift. P8 mentioned discussions as a way to prevent drift by ensuring awareness and commented on implemented architecture diverging from the intended architecture:

> *"Well, I don't know if it's unintentional because you always make some decision. [...] There is always a discussion."* (P8)

Tab. VI indicates that 68% of the survey participants reported that sharing information with others is a practice currently used to prevent and detect drift.

2) Practices to address drift:

TABLE IV
PRACTICES NAMED IN THE INTERVIEWS WITH ASSOCIATED SURVEY QUESTION(S). UNDERLINED PRACTICES ARE EXPLAINED IN DETAIL.

Current Practice	Description	Representative Quote	Survey Question
Practices related to detecting and preventing drift			
Established way of working	Following and relying on an established way of working to detect architectural drift (e.g., Scrum).	"As a developer, we will have some idea because we take part in the backlog refinement and the meetings with our product owner [...] so we definitely know that it is happening." (P3)	1 (Tab. VI)
Code reviews	Detecting and preventing drift when systematically reviewing other people's code.	"I rely on the manual eye and place a lot of emphasis on code reviews because that's where a lot [of deviations] can be caught." (P1)	1 (Tab. V)
Implementing code	Detecting drift while working with the code (e.g., implementing a new feature).	"You start seeing things that don't make sense. You see this either in an architectural document or you see it in the code: something does not make sense and does not correspond to what it should be." (P4)	2 (Tab. V)
Discussions	Discussions that involve collaborative conversations among team members can serve as a preventative practice.	"In a way, the discussions themselves are a practice. [...] I think the discussions in a way are there to, not only promote, but also to discourage certain drifts maybe." (P6)	3 (Tab. V)
Dedicated time	Dedicate time for maintenance and fixing technical debt as a way to prevent drift from occurring.	"Sometimes when we're building features, we take certain technical shortcuts and accumulate technical debt. The first thing we do afterwards is to actually fix the technical debt we created." (P2)	4 (Tab. V)
Integrated into release cycle	Preventing drift by checking the consistency between code and architecture as part of the release cycle.	"[Documentation] was an integral part of the release cycle itself. The documentation was good there." (P7)	5 (Tab. V)
Practices used to address drift			
Communication	Relates to relying on communication and reaching out to another person, either directly or during reoccurring meetings (e.g., daily standups).	"Normally it is during standups and basic communication: we just talk with each other." (P8)	"Do you use any practices to address drift in your team?"
Establishing practices			
Lack of practices	A lack of dedicated practices to detect, prevent and address architectural drift.	"I wouldn't say that there is a routine around it, more like conventions." (P7)	6 (Tab. V)
Part of the process	Drift is seen as a part of the development process.	"Discovering that it's not aligning anymore, it's something that evolves throughout the project. I don't think there is a single point that says 'Wow, this is outdated'." (P8)	3 & 4 (Tab. VI)

TABLE V
CURRENTLY USED PRACTICES - *survey respondents who selected a practice*.

Currently used practices	Total	Junior	Senior	Difference
1. "Reviewing other people's code (e.g., through code reviews)."	73%	68%	74%	-6pp
2. "Working with the code myself."	38%	32%	42%	-10pp
3. "Sharing information with others (e.g., in team discussions)."	68%	64%	71%	-7pp
4. "Dedicating time to fix accumulated technical debt."	56%	57%	55%	2pp
5. "Checking consistency between code and architecture as part of our release cycle."	19%	16%	21%	-5pp
6. "We do not have any dedicated practices."	11%	16%	8%	8pp

TABLE VI
ACCEPTABILITY AND AWARENESS - *survey respondents who agreed/strongly agreed with a statement*.

Current practices	Total	Junior	Senior	Difference
1. "We have sufficient practices in place to handle architectural drift."	64%	75%	57%	18pp
2. "I am aware when the code drifts from its intended architecture."	87%	83%	89%	-6pp
3. "Architectural drift is inevitable in software development."	65%	61%	68%	-7pp
4. "Architectural drift is acceptable."	54%	42%	62%	-20pp
5. "Architectural drift creates quality issues."	66%	68%	65%	3pp

a) Communication: More than 50% of the interviewees address drift by relying on communication, either directly or during reoccurring meetings (e.g., daily standups). P4 expressed: *"Depending on the situation I would talk to different people."* P1 also stressed the importance of intentions and stated: *"I speak with the developer who made that code change [...] and ask: 'Why did you choose to make this change?"*. Moreover, P2 stressed that the issue raised is not equal to a resolution of the problem:

> *"The first thing I do is just talk to the person [...] Maybe we'll come up with how we want to change things, or maybe we won't come to a conclusion."* (P2)

3) Establishing practices:

a) Lack of practices: 64% of the survey respondents reported that they have sufficient practices in place to handle drift. 11% of the survey respondents stated that they do not have any dedicated practices to detect or prevent drift. In some cases, our data still suggest that there is no dedicated practice in place to detect, prevent, or address drift. For example:

> *"[...] It is handled by spending more time on troubleshooting and reading code to understand it [...]. I don't think there is a good procedure for us; it just becomes a more challenging job."* (P9)

P5 stated: *"I don't know other than trying to establish standards and communicate them"*. P7 articulated: *"I wouldn't say that there is a routine around it, more like conventions."*

b) Part of the process: Our research indicates that drift is an integral part of the development process, and its detection is not feasible at a particular point in time, as it emerges through continuous discussions and communication. P6 emphasized that drift is gradual:

> *"We know when drift is happening because we are talking about what we would potentially want to do and discussing this as a team. [...] It's a gradual change [...]."* (P6)

Another example suggesting that developers view drift as gradual, also expresses that drift is first discovered when the code has become rigid:

> *"I think we are a part of it, it's not like we wake up one day and say: 'Okay, the architecture has drifted'. It's more like we are a part of it because in every sprint we have discussions."* (P11)

Tab. VI shows how 65% of the participants agree that drift is inevitable and 54% report that drift is acceptable.

Summary (RQ2): We found that a majority of participants consider themselves to be aware of drift. Reviewing code is

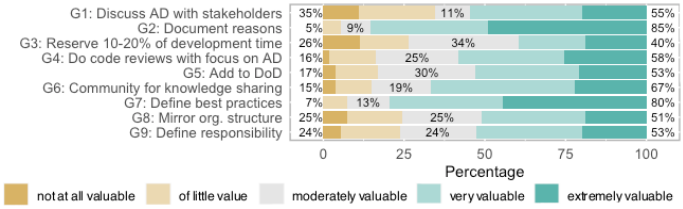

Fig. 3. Survey responses on the guidelines' value (n=63)

one of the main practices to detect drift. More seniors than juniors consider drift to be inevitable and acceptable.

C. Supporting Practices (RQ3)

We conceived supporting practices to mitigate drift, which are presented in Tab. VII. Based on our findings, we proposed 9 guidelines, which are shown in Fig. 3. It can be seen that the respondents consider it particularly valuable to document reasons for architectural decisions (G2) and define best practices (G7). The data concerning the difference between senior and junior developers can be found in the supplementary material[2]. Junior developers tended to consider the guidelines more valuable compared to senior developers. They rate G3 (reserve development time to counteract drift) and G5 (add drift-related aspects to the Definition of Done criteria) much higher than senior developers (26 and 20 percentage points, respectively), making these the two guidelines with the largest difference between senior and junior developers.

Below, we describe selected themes in further detail.

1) Defined responsibility: During the interviews, multiple participants brought up having defined responsibility as a supporting approach. 53% of the survey respondents regarded it very valuable or extremely valuable to define responsibilities for handling architectural drift. At the same time, opinions differ on who should be responsible for managing architectural drift. The responsibility for drift is mostly viewed as a team effort. As shown in Tab. VIII, our survey respondents stated that the responsibility for drift should lie with the software architects and the development team, while the product owner always remains at the bottom of the ranking. 47% of the senior developers thought that they should be responsible for drift, whereas 32% of the junior developers considered senior developers to be the ones who should be responsible for handling architectural drift. More junior developers than senior developers thought that the product owner should be responsible for handling architectural drift. P9 expanded on their view: *"I think it's not the responsibility of the individual developer, it's the responsibility of the solution manager"*. P2 thinks it is a team responsibility:

> *"It's very important that we as a team are collectively responsible [for the architecture] and say: 'This is our problem'."* (P2)

2) Set best practices: One of the challenges we found in the interviews was that organizations do not always have established mechanisms to detect, prevent, and manage architectural drift. In the survey results shown in Fig. 3, it can be seen that 77% of the survey respondents found it very valuable or extremely valuable to define best practices. Multiple participants mentioned the importance of organizational support for best practices and a need for common standards. P9 noted that they would like to have a more standardized way of working with architecture within their company: *"I would probably say that we should have one kind of standard on which we base our solutions."*

While P9 suggests that those standards should be decided by an architect, P2 instead suggests it should be decided by the teams:

> *"The teams work together to raise these common [practices] and set best practices [...]. Then, as the company changes over time, the team's architecture will change over time and then it will automatically ensure that best practices are updated over time."* (P2)

3) Foster good collaboration: Many interviewees mentioned good collaboration as important to mitigate drift. This includes building trust between different roles, continuous communication, and regular feedback. P2 stated:

> *"It's about continuously fostering a dialogue of understanding, helping one another, and primarily relying on each other."* (P2)

P1 stressed the importance of feedback and communication:

> *"Talk to each other in a team. Don't just get code reviews but you actually discuss the projects you're working on while you're doing it."* (P1)

Establishing a community of practice to share knowledge is considered valuable by the participants. The survey results show that 67% rated G6 (creating a community for knowledge sharing) as very or extremely valuable. Sharing knowledge about the rationales behind architectural decisions was considered particularly relevant. 85% of the respondents considered G2 (document reasons) very or extremely valuable.

4) Maintain reliable documentation: Maintaining reliable documentation is an important supporting practice for mitigating drift. Capturing information about why a decision was made and by whom is especially helpful for developers. 85% of the survey participants found G2 a valuable practice, making it the guideline that was considered most valuable, especially among junior developers. One developer stated:

> *"It would be good if we had some additional documentation on why we took certain decisions."* (P6)

Having documentation as an integrated part of the release cycle was brought up as an example of a practice to maintain good and complete documentation. According to our findings, 53% agree that G5 *"Adding architectural concerns to the Definition of Done criteria"* would be very valuable or extremely valuable.

Summary (RQ3): We conclude that especially documenting reasons behind architectural decisions (G2) and defining best practices (G7) were considered valuable by the participants. Juniors consider it more useful than seniors to add

TABLE VII
SUPPORTING PRACTICES WITH QUOTES AND RELATED GUIDELINES IN THE SURVEY. UNDERLINED PRACTICES ARE EXPLAINED IN DETAIL.

Practice	Description	Representative Quote	Survey
Defined responsibility	Refers to how clearly defined responsibilities can help organizations to mitigate drift.	"It's very important that we as a team are collectively responsible [for the architecture] and say: 'This is our problem'." (P2)	G9
Set best practices	Refers to having common best practices for developers on an organizational level (e.g., best practices set by developers or communicated to developers).	"I would probably say that we should have one kind of standard on which we base our solutions. If we know that we have a few different solutions that are similar, we should build them in a similar way so that we have different cases that can be reused and therefore can be managed." (P9)	G7
Increased technical competency	Refers to increasing technical competency of business stakeholders.	"I usually want the [business] stakeholder to have higher technical competence." (P7)	G1
Mirror communication structure to intended architecture	Refers to acknowledging that the organizational structure (e.g., the setup of teams and communication paths) impacts how the architecture is implemented and can be used deliberately by practitioners to their advantage.	"[...] you think about how we want our architecture and set up the teams accordingly. That's a very interesting thing that I think more companies should follow." (P2)	G8
Foster good collaboration	Refers to good collaboration (e.g., trust between different roles, feedback, and continuous communication and discussions) as important to mitigate drift.	"To actually talk to each other in a team. That you don't just get code reviews but you actually discuss the projects you're working on while you're doing it. Because someone else's opinion can be very useful." (P1)	G6 & G4
Maintain reliable documentation	Refers to documentation as an important practice to mitigate drift. Capturing information about why a decision was made and by whom is especially helpful for developers.	"I would honestly want to have more documentation on it. [...] But anyone who gets onboarded might not understand how we got to this point, and it would be good if we had some added documentation on why we took certain decisions." (P6)	G2

TABLE VIII
RESPONSIBILITY - *survey respondents' role selections*.

Statement	Total	Junior	Senior	Difference
Who is currently responsible for handling architectural drift?				
1. The individual developer	38%	40%	37%	3pp
2. The development team	57%	56%	58%	-2pp
3. Senior developers	33%	28%	37%	-9pp
4. Product owner	3%	8%	0%	8pp
5. The software architect(s)	52%	56%	50%	6pp
Who should be responsible for handling architectural drift?				
1. The individual developer	36%	40%	34%	6pp
2. The development team	70%	72%	68%	4pp
3. Senior developers	41%	32%	47%	-15pp
4. Product owner	19%	24%	16%	8pp
5. The software architect(s)	71%	68%	74%	-6pp

drift-related concerns to the Definition of Done criteria and to reserve time to identify and counteract drift.

V. DISCUSSION

This study sheds light on architectural drift, challenges connected to it, and current and potential ways to detect, mitigate, and address it. Some of our insights confirm previous findings. That is not surprising and not the goal of empirical research, where we aim to provide evidence on the practices used in industry. To the best of our knowledge, no empirical study has been published that focuses on the developers' perspective on architectural drift.

a) Prioritization and the role of agility: A majority of the survey respondents (66%) stated that architectural drift creates quality issues. Fewer junior developers than senior developers stated that drift is acceptable. The participants of our study consider it important to keep the code aligned with the intended architecture. Overall, only 41% of the survey respondents stated that they deviated from the intended architecture due to a lack of time. To some extent, this finding contradicts previous research where a lack of time, resources, and knowledge are given as reasons for architectural inconsistencies [10]. Related work suggests that using iterative software development processes can contribute to architectural drift [9], [18]. While we did not find that agile processes are problematic per se, we did see issues connected to the difficulty of prioritizing architectural work over business-driving objectives. Instead of framing issues as being due to a lack of time, our respondents stressed the importance of prioritization.

b) Existence of architectural documentation: While all participants in this study had some mechanisms to capture the intended architecture, many organizations do not have formal architectural descriptions. Descriptions can be whiteboard pictures, documents, or models. According to our findings, even if the description does not exist, companies tend to have an original intention. In that case, drift may still exist and is visible in the implementation that deviates from this intention over time.

c) Tools for architectural drift: Despite numerous proposed formal approaches and tools [9], [12], [2] to achieve architectural consistency, we found that most practices used in the industry are informal. While a previous study on architectural consistency found that some practitioners use tools like SonarQube [3], we did not find a company using similar tools. Related to architectural erosion, previous research [9] suggests that the detection of deviations is particularly challenging.

d) Awareness: Our findings also indicate that there currently is a notable risk associated with the practices that are used to detect drift, as they heavily depend on the specific developer's experience and level of awareness. While previous work suggested that developers tend not to be aware of the architectural impact of changes in code reviews [28], the majority (88%) of our respondents agreed or strongly agreed with the statement *"I am aware when the code drifts from its intended architecture"*. This finding indicates that it is difficult to state how aware one is of potentially unknown things like drift. Moreover, some developers might view drift as gradual and a process that they are a part of.

We also found that although practitioners regard it as beneficial to document reasons behind decisions, few participants follow that practice. This might be due to the difficulty of formulating rationales, which is why the importance of this practice needs to be stressed.

e) Differences between junior and senior developers: We found that the way junior and senior developers reason about architectural drift is quite different. We found that although developers accept the inevitability of drift, especially junior

developers appreciate practices that enable them to mitigate it. Our proposed guidelines can aid practitioners in this regard. Juniors tend to rate our guidelines higher than seniors.

f) Practices to deal with drift: Several practices were suggested and are currently followed to deal with architectural drift. It should be noted that these practices are rather general and high-level. Given that dealing with software architecture is very context- and system-specific, they need to be tailored to individual organizations. We acknowledge that their reported usefulness in this paper is based on the perceptions of our 11 interviewees and 63 survey respondents and that long-term studies are needed to better understand how drift can be detected, mitigated, and addressed in practice. The guidelines rated as the most useful are documenting reasons behind architectural decisions (G2) and defining best practices on an organizational level (G7). These findings indicate that practitioners see a need to establish organizational and documentation practices for drift, rather than to only focus on technical tools and solutions [6]. Some of our guidelines can help to support developers to become more aware of an evolving architecture over time. The guidelines to document rationales and to define best practices might be a first step, so that drift can be addressed and evolution can happen in a controlled way.

A. Threats to Validity

In the following, we discuss threats to validity [29], [30].

1) Construct validity: We strove to ensure that the construct of architectural drift was well-defined. We achieved this by conducting a thorough review of the literature. The interview guide included questions regarding the participants' own definition of drift and the interviewers also provided a definition of drift. Similarly, the survey contained a definition of drift. We provided the definition to avoid confusion and establish a common understanding of the concept.

2) External validity: The purpose of this mixed-methods study is to identify a localized truth and not to search for generalizable results. The sample of participants is still a crucial factor. To improve external validity, we included participants working across diverse software companies and domains, and varying levels of experience.

3) Reliability: To ensure that the study can be replicated, we uploaded supplementary material[2], such as the interview guide. We expect that by using this material, it will be possible for other researchers to replicate the study. We described the decisions and steps taken during the data collection and analysis to ensure transparency.

In qualitative studies like this one, there is a threat that the questions in the interviews may not have been well-phrased or we as interviewers might have unconsciously influenced the interviewee's answers. To mitigate this threat, we aimed to create short and clear questions. We also ran two pilot interviews to capture potential issues.

To minimize bias in the analysis, the first two authors conducted the inductive phase of the thematic analysis individually. Then, they compared their codes and notes and discussed any disagreements. The codebook with descriptions of all themes and codes is provided in the supplementary material. We aimed to work consistently with the codebook throughout the analysis. In a final iteration of the coding process, the researchers went through the data again and double-checked that the codes were correctly and consistently applied. To reduce the potential influence of personal beliefs on data interpretation, we employed triangulation, by validating data collected during the interviews with a survey [31].

VI. Conclusions and Future Work

Our findings provide insights into how developers perceive drift. They consider it important to prioritize architectural consistency and talk about architectural decisions. Junior developers see a higher value in documentation, whereas senior developers consider communication and discussions more central. Based on our analysis of the current state, we explored several practices that can help developers to identify and counteract drift. Given the complexity of drift, we suggest that practitioners consider drift from multiple perspectives, both technical and non-technical. There exist different opinions regarding who is and who should be responsible for handling architectural drift, with the majority of the respondents stating that the responsibility should lie with the development team and software architect(s).

Our study sheds light on the many challenges and perceptions that exist around architectural drift. We believe our study can be a starting point to provide greater awareness and useful tools to prevent and combat architectural drift, both for software developers and architects. Developers often use informal practices to manage drift. Our research highlights the need for evaluating strategies, emphasizing collaboration, clarifying responsibilities, and enhancing technical expertise to tackle drift successfully.

Researchers can build upon these findings and identify how the guidelines can be customized to specific contexts. Future directions are to investigate lightweight mechanisms to help practitioners to document reasons for architectural decisions, share knowledge, and do code reviews with a focus on drift.

We would like to further refine our proposed guidelines and conduct a longitudinal study on the effects of applying them in industrial settings. One interviewee stressed that tools to create documentation automatically would be beneficial. At the same time, there are no established state-of-the-practice tools that help companies to mitigate architectural drift. It would be interesting to better understand what the reasons are and how tools could address practitioners' needs in the future.

Acknowledgments

We would like to thank all participants for their invaluable support. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and by Stiftelsen C.M. Lerici Foundation.

REFERENCES

[1] M. Nagl, "The architecture is the center of the software development process," RWTH Aachen, Tech. Rep. AIB-2021-08, Nov. 2021, accessed on February 26, 2024. [Online]. Available: http://publications.rwth-aachen.de/record/835237/files/835237.pdf

[2] L. De Silva and D. Balasubramaniam, "Controlling software architecture erosion: A survey," *Journal of Systems and Software*, vol. 85, no. 1, pp. 132–151, 2012.

[3] N. Ali, S. Baker, R. O'Crowley, S. Herold, and J. Buckley, "Architecture consistency: State of the practice, challenges and requirements," *Empirical Software Engineering*, vol. 23, pp. 224–258, 2018.

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

[5] L. Hochstein and M. Lindvall, "Combating architectural degeneration: a survey," *Information and Software Technology*, vol. 47, no. 10, pp. 643–656, 2005.

[6] S. Herold, M. Blom, and J. Buckley, "Evidence in architecture degradation and consistency checking research: preliminary results from a literature review," in *Proccedings of the 10th European Conference on Software Architecture Workshops*, 2016, pp. 1–7.

[7] J. Buckley, S. Mooney, J. Rosik, and N. Ali, "JITTAC: A just-in-time tool for architectural consistency," in *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, 2013, pp. 1291–1294.

[8] D. Ganesan, T. Keuler, and Y. Nishimura, "Architecture compliance checking at runtime: An industry experience report," in *Proceedings of the 8th International Conference on Quality Software*, 2008, pp. 347–356.

[9] R. Li, P. Liang, M. Soliman, and P. Avgeriou, "Understanding software architecture erosion: A systematic mapping study," *Journal of Software: Evolution and Process*, vol. 34, no. 3, p. e2423, 2022.

[10] R. Wohlrab, U. Eliasson, P. Pelliccione, and R. Heldal, "Improving the consistency and usefulness of architecture descriptions: Guidelines for architects," in *Proceedings of the IEEE International Conference on Software Architecture (ICSA'19)*. IEEE, 2019, pp. 151–160.

[11] R. Li, P. Liang, M. Soliman, and P. Avgeriou, "Understanding architecture erosion: The practitioners' perceptive," in *Proceedings of the IEEE/ACM 29th International Conference on Program Comprehension (ICPC'21)*. IEEE, 2021, pp. 311–322.

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

[13] N. B. Moe, D. Šmite, M. Paasivaara, and C. Lassenius, "Finding the sweet spot for organizational control and team autonomy in large-scale agile software development," *Empirical Software Engineering*, vol. 26, no. 5, p. 101, 2021.

[14] C. Wilson, "Chapter 2 - semi-structured interviews," in *Interview Techniques for UX Practitioners*, C. Wilson, Ed. Boston: Morgan Kaufmann, 2014, pp. 23–41.

[15] A. N. Ghazi, K. Petersen, S. S. V. R. Reddy, and H. Nekkanti, "Survey research in software engineering: Problems and mitigation strategies," *IEEE Access*, vol. 7, pp. 24 703–24 718, 2019.

[16] J. B. Tran and R. C. Holt, "Forward and reverse repair of software architecture," in *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research*, S. A. MacKay and J. H. Johnson, Eds. IBM, 1999, p. 12.

[17] J. Rosik, A. Le Gear, J. Buckley, M. A. Babar, and D. Connolly, "Assessing architectural drift in commercial software development: a case study," *Journal of Software: Practice and Experience*, vol. 41, no. 1, pp. 63–86, 2011.

[18] J. van Gurp and J. Bosch, "Design erosion: Problems and causes," *Journal of Systems and Software*, vol. 61, no. 2, pp. 105–119, Mar. 2002.

[19] E. Whiting and S. Andrews, "Drift and erosion in software architecture: Summary and prevention strategies," in *Proceedings of the 4th International Conference on Information System and Data Mining*, ser. ICISDM 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 132–138.

[20] S. Baltes and P. Ralph, "Sampling in software engineering research: A critical review and guidelines," *Empirical Software Engineering*, vol. 27, no. 4, p. 94, 2022.

[21] J. Robinson, *Likert Scale*. Dordrecht: Springer Netherlands, 2014, pp. 3620–3621.

[22] U. N. (UN). (2024) Guidelines for gender-inclusive language in English. Https://www.un.org/en/gender-inclusive-language/guidelines.shtml. Accessed on February 26, 2024.

[23] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[24] V. Braun and V. Clarke, *Thematic analysis.* American Psychological Association, 2012.

[25] ——, "Reflecting on reflexive thematic analysis," *Qualitative Research in Sport, Exercise and Health*, vol. 11, no. 4, pp. 589–597, 2019.

[26] C. de O. Melo, C. Santana, and F. Kon, "Developers motivation in agile teams," in *Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2012, pp. 376–383.

[27] A. Ju, H. Sajnani, S. Kelly, and K. Herzig, "A case study of onboarding in software teams: Tasks and strategies," in *Proceedings of the 43rd International Conference on Software Engineering (ICSE'21)*. IEEE, 2021, pp. 613–623.

[28] M. Paixao, J. Krinke, D. Han, C. Ragkhitwetsagul, and M. Harman, "Are developers aware of the architectural impact of their changes?" in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE'17)*, Oct 2017, pp. 95–105.

[29] R. K. Yin, "Designing case studies," *Qualitative research methods*, vol. 5, no. 14, pp. 359–386, 2003.

[30] F. Shull, J. Singer, and D. I. Sjøberg, *Guide to Advanced Empirical Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2007.

[31] M. Kasunic, "Designing an effective survey," Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, Tech. Rep., 2005.