The beauty of software architecture

1st Céline Madeleine Aldenhoven
Software Engineering (Elite Graduate Program)
Technical University of Munich,
Ludwig-Maximilian-University of Munich,
University of Augsburg
Munich and Augsburg, Germany
celine.aldenhoven@tum.de

2nd Ralf Sascha Engelschall
msg Research
msg systems AG
Ismaning, Germany
ralf.engelschall@msg.group

***Abstract*—The concept of beauty has fascinated humans since the beginning of their existence. To inspire research questions on the beauty of software architecture, we use the progress in other areas, like civil architecture and plastic surgery. Our most important findings are that practitioners view software architecture as beautiful if it reduces their effort to work with it and that the beauty of software architecture greatly impacts the happiness and motivation of the developers working on it. Overall, this research paper gives insights into the definition of beautiful software architecture in practice and explains its importance. Additionally, we propose ways to create beautiful software architecture and teach it. The focus on beauty in software architecture is essential in mid- to large-sized projects to increase the developers' happiness and, therefore, the quality of the product.**

***Index Terms*—Software Architectures, Design, Health implications, Education**

I. Introduction

Throughout the centuries, humans have been mesmerized by the concept of beauty. Many of the works of Plato, Da Vinci, Kant, and others attempt to define it. Efforts to measure beauty are the golden ratio in aesthetics or the "perfect face" ratios in plastic surgery, for instance. Various aspects of beauty, have received enormous attention. This attention has contributed to significant progress in establishing criteria for defining it from different points of view [1].

In our experience, a similar concept exists in software architecture. Analyzing different types, we find some are perceived as beautiful. Few literature sources have started to engage in the topic of beautiful software architecture [2], [3], [4], but more research is necessary to investigate it further.

Beauty is a topic studied in several disciplines, like philosophy, plastic surgery, and civil architecture [5], [6], [7]. Studies in anthropology, for instance, demonstrated that people of all cultures experience beauty. However, the precise definition of what constitutes beauty varies from culture to culture [8]. Moreover, all individuals can perceive beauty. For instance, according to the research of Dayan et al., people who are blind can still perceive beauty [9]. Looking at software architecture, what do developers and software architects view as beautiful in software architecture, and is it an extremely subjective matter, or is there a consensus among them? This question is one of the research questions we are answering with this research.

This research was possible thanks to the funding of msg systems AG.

Several studies have shown that people perceived as beautiful are happier humans. Statistics show that better-looking people have better education, health, and chances to be hired for a job [10]. Characteristics like success, dominance, and extroversion are positive preconceptions that we have of beautiful people [11], [12], [13]. Beauty stimulates pleasure in the person perceiving it, which is why we feel better when surrounded by beautiful things and people [14]. These findings inspired other research questions that we answer in this paper. Has the beauty of software architecture affected the emotions and mental health of the developers and software architects working on it? Moreover, what preconceptions do they have of software when they perceive its architecture as beautiful?

Only some studies have researched the topic of beauty in software architecture. The previous works have mainly tried to define it theoretically. A profound examination of the professional's view on beauty in software architecture, its correlation with the definitions in literature, and the examination of its significance in the professional workplace still have to occur. Therefore this study describes the concept of beauty in software architecture for practitioners and explains its importance for development projects. To apply the insights from our study in the corporate world, we also propose ways to create beautiful software architecture and concepts on how to teach it to the next generation.

II. Related Work

Other disciplines have inspired several computer science concepts. One example would be the work of Christopher Alexander, an architect, who inspired some patterns for software architecture [4], [3]. Therefore, were inspired by civil architecture, philosophy, plastic surgery, and other fields' papers in addition to computer science-related papers.

A. Civil Architecture Research

The beauty and the aesthetics of a building are two different termini. Beauty describes the visual and intellectual pleasure that a building brings its viewers, while aesthetics describes its looks [15]. Being delighted by beautiful structures is what makes them different from others. Even though beauty brings pleasure, admitting its worth entails some shame in the civil architecture criticism environment [15].

Although the architect must be able to "suppose that what he or she regards as beautiful would have universal assent", beauty is frequently referred to as entirely subjective [7]. This shared understanding might also exist in the perception of beauty in software architecture. A statement from the same study that categorizes mathematical beauty as objective backs up this idea. It is objective because mathematical beauty must "obey the logical deductive rules of the brain, rules that are common to all humans, irrespective of cultural and ethnic differences" [7]. Regardless of all other requirements placed on architectural design, beauty must be a central element. Its experience contributes to the health of the individual and thus to the well-being of society. This makes beauty in civil architecture not a luxury but a necessity [7].

Several researchers promote beauty in civil architecture as functionalism. Here, the form of a building should follow its function and beauty comes from the fulfillment of this function [16], [17]. According to Vitruvius, the three main requirements for civil architecture are strength, usefulness, and beauty. All three categories must be fulfilled equally [18]. As software and civil architecture have some intended functionality in their design, beauty in software architecture might be defined similarly to the buildings'. One example of a concrete theory on achieving beauty in civil architecture is via the golden ratio [17].

Christopher Alexander's research in civil architecture has inspired the creation of software patterns [19], [20], [4], [3]. Therefore, his theories on beauty could be of particular interest in software architecture. For Alexander, "good design is defined in terms of the absence of faults or misfits" [21]. Each one of us can create something beautiful. For an architect, this talent lies in their ability to observe correctly and deeply [22]. This finding implies that experience can teach beauty.

B. Research in Other Fields

In addition to the research works described in the introduction, many more studies have dealt with beauty. According to the definition of the philosopher Immanuel Kant, beauty is what delights without looking at its purpose [5]. Beauty is something we can "feel" and requires thought, whereas non-beautiful pleasures do not [23]. A paper about the experimental psychology of beauty states that it is "widely admitted that, in general, the enjoyment of beauty is accompanied by pleasure" [24]. As the "human mind prefers to focus on aesthetically pleasant objects rather than unpleasant ones" [1], [25], it should be examined whether developers prefer to work with beautiful software architecture.

The research of Sisti et al. has described beauty from a plastic surgery point of view [1]. Their paper states that symmetry is essential in facial beauty [26], [27]. Scars and other factors that undermine facial symmetry negatively influence the perceived facial beauty [28]. Many scientists have tried to find the ideal proportions of beauty, and many of these ideal aesthetic dimensions come from ancient Greek and European Renaissance art, like the one of Leonardo Da Vinci [29], [30], [31]. Leonardo Da Vinci painted the Vitruvian Man to demonstrate the ideal human body proportions, inspired by the definitions of the ancient Roman architect Vitruvius. All these efforts to evaluate beauty by mathematical ratios and parameters do not pay attention to cultural and ethical dissimilarities [1], [32], [33], [34], [35]. According to Sisti et al., "cultural differences play an essential role in these measurements and may vary by ethnicity [1], [30], [31], [36]. Wang et al. [37] demonstrated that North American white patients have a greater ratio of mouth width to nose width, while Han Chinese patients have a wider nose and a narrower mouth [37]. In black individuals, though, the nose has a greater width in comparison to white individuals [38]" [1]. To evaluate beauty, one has to contemplate the importance of cultural and ethical differences as well [35], [39], [40], [41].

C. Computer Science Research

When Le et al. studied architectural decay in systems using the example of open-source software, they found that architectural shortcomings have noticeable negative consequences in the form of implementation problems and code commits. These require increased maintenance over the life of a system [42]. Therefore it would be interesting to study if beautiful software architecture keeps maintenance low.

An architectural violation is one of the most frequently identified symptoms of architectural erosion. This finding emphasizes that maintenance of the software architecture affects the system's quality over time [43]. Therefore we investigate whether a beautiful software architecture could increase the developers' motivation to maintain it, which helps maintain the quality of the whole software system over time.

Spinellis and Gousios wrote a book that discusses beauty in software design and architecture. Overall, they define beautiful software architecture by its utility, buildability, well-defined structure that supports incremental construction, well-defined and modular interfaces, inherent testability, maintainability and flexibility, and features that delight developers and testers [2].

Even though beauty is referred to as subjective sometimes, some architectural patterns can be reused for specific problems and help achieve an elegant and, therefore, often beautiful solution. For instance, the software architecture of different accounting systems is remarkably similar due to the stability of accounting basics over time [44]. Some papers state that clear objective criteria for beauty, like reliability, extendability, and reusability, exist [45], [46]. Findings like these support our aim to search for similarities in the perceptions of beautiful software architecture from developers and software architects.

For Fujino, the beauty of software architecture is one of its most critical quality characteristics. For Fujino, "aesthetic and engineering aspects of software can coexist to create a harmony of beauty and rationale" [4]. Fujino is convinced that carefully examining ancient architectural methods will assist us in finding patterns that will realize the harmony of beauty and rationality. The first step is to refocus on Christoper

Alexander's original works, which emphasize aesthetic concerns and utility [4].

According to the research of Appleton, "objective beauty" exists and is closely linked to the presence of symmetries. However, the paper also states that "if the symmetries are too pure (too perfect or exact), it seems to be less desirable than if slight irregularities and imperfections exist" [3].

Defining software architecture is complicated because "software development does not have a unique architectural level of design" [47]. Abstracting the software design and architecture of the system from all its details is very difficult [48]. We use different models (conceptual, static, dynamic) to describe one software architecture. Therefore, software development can be called a "model building discipline" [47]. "The only way we can cross these gaps (model vs. implementation) is through mental processes, such as abstraction, which are a part of the mental faculties of all of us. However, the problem is that they are subjective to the person performing the transformation, resulting in different results for different developers" [47].

III. Methods

To study beauty in software architecture, we proposed four Research Questions (RQs):
RQ1. What are the discrepancies between practitioners' and previously published theoretical definitions of beautiful software architecture?
RQ2. What is the importance of beauty in software architecture?
RQ3. How can we create beautiful software architecture?
RQ4. How can we teach the intuition for beauty in software architecture?

A. Study Design

To answer our research questions, we conducted a survey questionnaire to ask practitioners about their experiences and perceptions, an interview with expert practitioners to discuss some more detailed questions with experts, and utilized reference literature for theoretical definitions. After thoroughly reviewing the related work, we created the survey questionnaire and the interview structure. All the materials used in this study and the anonymized survey and interview results produced in this context can be requested from the authors in case that the provided link does not work anymore[1].

To compare our findings from practice to a definition in literature, we used the book "Beautiful architecture: leading thinkers reveal the hidden beauty in software design" by Spinellis and Gousios [2]. This book is often quoted in the context of beauty in software architecture.

1) The structure of the survey questionnaire: We have conducted a descriptive survey, following the guidelines proposed by Kitchenham and Pfleeger [49], [50]. The participants filled out the survey online. Our survey is composed of 16 questions (Q1-Q16). Fourteen have been mandatory, two voluntary, and depend on personal experience. According to our inclusion criterion, we discarded data sets from people with insufficient development experience afterward.

[1]Interview and survey questions and data: https://drive.google.com/drive/folders/1zqeUfEzia5vdOIUl5Bs5j1nPbxLgU52J?usp=sharing

Our inclusion and exclusion criteria for this study:

- *Inclusion criterion:* The participant has more than one year of experience in software development.
- *Exclusion criterion:* The response does not answer the question.

The first two questions of the survey asked for information about how many years of professional experience the participant has as a software developer and as a software architect. This information enabled us to get more insights into how detailed the participants have thought about software architecture already. To ensure the privacy of our participants, these are the only personal related pieces of information we gathered. This data helped us to assess the participant's qualifications for the survey based on the inclusion and exclusion criteria.

The rest of the survey contains open-ended questions and close-ended questions. Yes/No answers and rating scale questions [51] have been used as close-ended question types. We asked the rating scale questions on a scale of 0-9. We added the option "sometimes" to the yes/no answering possibilities of the question that asked whether a beautiful software architecture makes software superior to one without. Here we reasoned that only allowing yes/no answers on this question would impose too many restrictions on the participants' answers and therefore bias our results. Because the option "yes" implies that software with beautiful architecture is always superior and the option "no" implies that it is never superior, a third option was needed that allowed the participants to state that superiority is only sometimes the case.

2) The structure of the interviews: We have conducted interviews to find out how beauty in software architecture can be taught to the next generation. All our interview participants are experienced architects from a single project organization [52].

We have conducted semi-structured interviews to collect industry expertise on both planned-ahead and unforeseen topics [53], [54], [55]. Furthermore, we utilized the checklists provided by Ralph et al. [56] that help to ensure scientific rigor for empirical studies in software engineering.

The interviews were conducted via video calls. All eight industry experts answered three open-ended questions about how they have developed their sense of beauty in software architecture, how it might have been possible to develop this sense earlier, and how we could teach the concept of beauty in software architecture to the next generation. Their ideas were discussed, according to the rules of semi-structured interviews.

3) Target population recruitment and sampling: We used different channels to reach our target population of developers and software architects. As this research was sponsored by industry, we contacted senior software architects at this company and invited them to participate in the survey and the interviews. These seniors can hardly be reached without an industry contact or similar. Additionally, the invitation to participate in our survey has been sent to university contacts in our target population. Furthermore, we contacted GitHub developers by collecting the email addresses in the user profiles of the contributors to the most trending repositories, like other studies [50], [57], [58], [59]. Lastly, we increased the number of participants in the survey by sending out reminder emails and by asking our potential participants to invite other people in our target group, according to the snowballing sampling method [60].

4) Validation and evaluation of the survey and interview design: All authors have thoroughly reviewed all parts of the survey and interview protocol before conducting the survey online for data collection and the interviews via video call.

Moreover, we also performed a pilot survey by sending invitations to five participants. Among these pilot testers, three were from academia, with thorough knowledge of survey design, and two were from the private sector. All five participants are actively involved in software development with 3 to 25 years of experience. The initial test of the interview questions was conducted via the pilot study with one participant with 25 years of experience in development and 15 years of experience as a software architect.

Several factors that could influence the outcomes of this study have been examined during the pilot survey and interview. These factors include the comprehensibility of the survey and interview questions and the fit of the survey and interview questions for our research questions. Based on the responses from the pilot survey study and the pilot interview, we refined one survey question that asked whether the participant has ever tried to create beautiful software architecture or demanded it as a customer before to avoid a misunderstanding that occurred in the pilot survey study. Nothing had to be adapted in the interview protocol, after the pilot interview study.

5) Filtering valid responses from the survey and interview: On the survey invitations, we received 66 responses. Five responses have been excluded according to our inclusion and exclusion criteria. For the interviews, eight industry experts took their time to participate. Overall, we had 61 valid responses from participants in the survey and eight valid expert responses from participants in the interviews.

The survey had two questions that were not mandatory because they required previous development experience. We expected that the 61 valid answers who stated they have at least one year of experience would answer these questions. This was not the case. For the question related to working with beautiful software architecture, we got 54 answers. For the one that asked about working with ugly software architecture, we received 53 answers from 61 participants.

B. Survey and Interview Data Analysis

The responses to our survey and interview questions have been analyzed using two approaches. The open-ended questions were analyzed using open coding and constant comparison techniques from Grounded Theory [61]. For the closed-ended questions, we used the descriptive statistics approach [62], to understand each response's occurrences. The interviews have been interpreted qualitatively, while the survey has been examined quantitatively, except for the question of creating beautiful software architecture. This question was examined qualitatively as well, as this was a better fit for the related RQ4.

To support the analysis process, we used a qualitative data analysis tool (MAXQDA2022) [53], [50]. Firstly, all main ideas in each response have been summarized in a code. Afterward, we used our analysis tool's "creative coding" interface to use constant comparison as effectively as possible to merge codes with similar semantic meanings into one. Finally, similar codes have been grouped into high-level categories. On some rare occasions, we had to code an answer as "excluded" when the participant's response did not answer our question. For instance, in the question about the definition of software architecture, the response "understandable, self-explanatory, comprehensible, robust, sustainable, etc." has been coded as "excluded" because it describes quality criteria and does not define software architecture.

IV. Results

A. Demographics and Software Architecture Definition

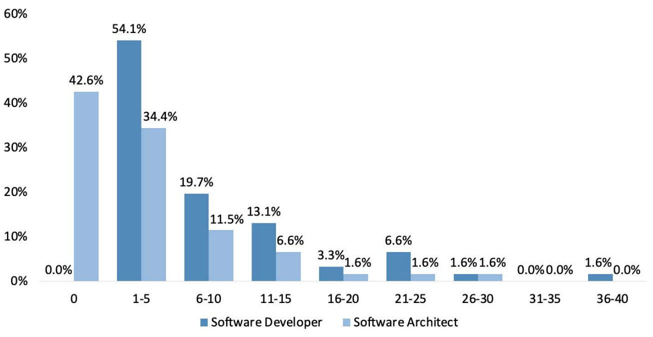

Fig. 1. Years of professional experience from valid survey responses.

Our survey had 66 participants and 61 valid responses, according to our inclusion and exclusion criteria. Additionally, we had eight valid interviews with expert participants. Valid datasets include surveys that have been filled out partially valid. The answers to the questions that need to be excluded according to the exclusion criterion have been coded as "excluded" and left out of the analysis of that particular question.

For the survey, around 46% of survey participants have more than five years of experience in software development, while nearly 10% have more than 20 years of experience (Q1). Furthermore, approximately 57% of the participants have at least one year of professional experience as a software architect. Moreover, approximately 23% have more than five years of experience as software architects, while around 3% have more than 20 years of experience (Q2). These findings can be examined in detail in Figure 1.

In the interviews, more than 75% of the 8 participants have more than five years of experience in software development, while nearly 37.5% have more than 20 years of experience. Furthermore, all participants have at least one year of professional experience as software architects. Moreover, 75% have more than five years of experience as software architects, while around 12.5% have more than 20 years of experience.

In literature, software architecture is defined as a minimal abstract view, a deep structure, and a description of all components of a system and its interactions with each other and external objects. Therefore, software architecture is, according to literature, the structure and behavior of a system and the related descriptions of those [2], [63], [64]

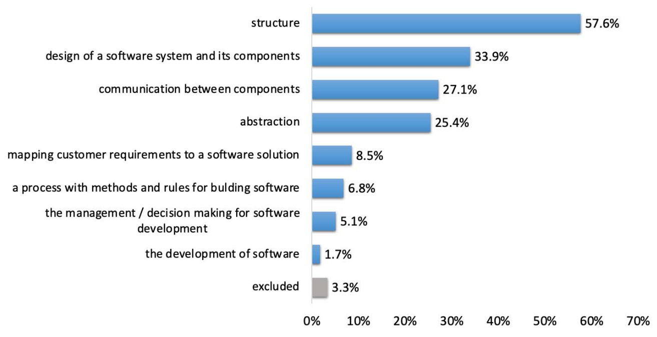

Fig. 2. What is understood by software architecture in practice?

To find out what is understood by the term software architecture in practice, we asked that question in our survey (Q3). For practitioners, software architecture refers to the system's structure, design, and communication among its components. The answers of the participants can be found in Figure 2.

Overall, the answers from practitioners are very similar to the definition of software architecture in literature. The differences were that the literature describes software architecture also as the diagrams and other descriptions of the actual architecture of a system. This was not directly reflected in the answers of the practitioners, even though it can be argued whether some participants might have indirectly included diagrams and other descriptions in their answers about 'design', 'structure' and 'abstraction'.

B. RQ1. What are the discrepancies between practitioners' and previously published theoretical definitions of beautiful software architecture?

According to Spinellis and Gousios, beautiful software architectures show some universal principles: 1) Beautiful software architecture has one fact in one place, meaning duplication shall be avoided. This principle also applies to behavior. 2) The architecture has to support the propagation of data or behavior into many places where they may be used efficiently. 3) Beautiful architecture is beautiful at runtime, as well as at construction time. 4) Beautiful architecture employs a "minimal set of mechanisms that satisfy the requirements of the whole". 5) Beautiful architectures are based on use cases and one functionality at a time. 6) The architecture considers the direction of likely growth and accounts for it. 7) Beautiful architecture resits entropy by supporting maintenance while preserving the architecture over time.

Spinellis and Gousios state that beautiful software architecture is defined by its utility, buildability, a well-defined structure that supports incremental construction, well-defined and modular interfaces, inherent testability, maintainability, flexibility, and features that delight developers and testers. One example of a delightful feature is conceptual integrity [2]. Additionally, beautiful software architecture can be defined by its "commercial success, influence on other product line architectures (others have 'borrowed, copied, or stolen' from the architecture), and sufficient documentation" [2].

We asked preliminary questions to get our study participants to start thinking about beauty in software architecture (Q4, Q5). The results were that among our participants, 91.8% think that software architecture can be beautiful, and one has already appeared beautiful to 85.2%.

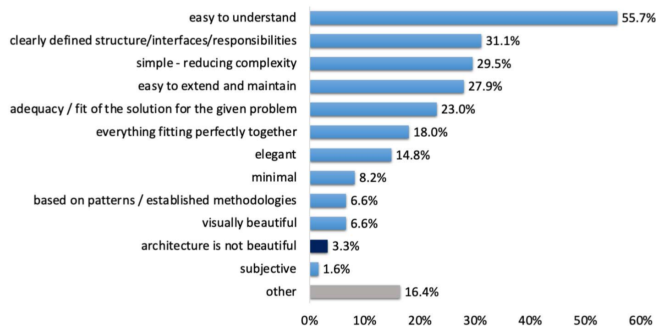

Fig. 3. What is understood by beautiful software architecture in practice?

As we wanted to discover whether there is some common understanding of the perception of beauty among developers and software architects, we asked our survey participants how they define beautiful software architecture (Q6). The results of this question can be found in Figure 3. During further examination, we discovered that 88.5% of responses mentioned a feature that makes the software easier to develop. We grouped the codes "easy to understand", "clearly defined structure/interfaces/responsibilities", "simple - reducing complexity", and "easy to extend and maintain" in this analysis. Beauty was described as reducing complexity by 63.9% of participants. These answers include the codes "clearly defined structure/interfaces/responsibilities", "minimal", and "simple - reducing complexity".

Most practitioners perceive beauty in software architecture as easy to understand, flexible and maintainable, reducing complexity, clearly defined structure/interfaces/responsibilities, and adequacy. These characteristics are also mentioned in the literature but emphasized more in practice. We infer that practitioners perceive architecture as beautiful if it reduces their effort to work with it. The primary goals are reducing complexity and making the software easier to use.

Key Findings of RQ1
Finding 1: In 88.5% of the responses, the participants stated that beautiful software architecture makes software easier (to understand, maintain, etc.).
Finding 2: Almost two thirds of answers indicated that beautiful software architecture is reducing complexity.
Finding 3: Deriving from findings 1 and 2, practitioners view software architecture as beautiful when it reduces their effort to work with it.

C. RQ2. What is the importance of beauty in software architecture?

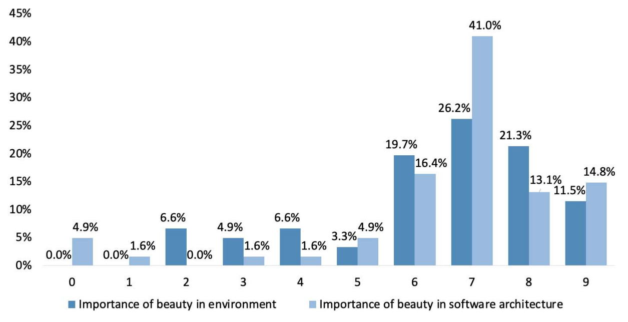

Fig. 4. How important is beauty for practicioners

We wanted to determine how vital our study participants find beauty in their everyday life (e.g., being surrounded by a beautiful environment) and in software architecture, with Q7 and Q8 in our survey. The results can be found in Figure 4. While 81.8% stated that the importance of beauty is at least a 5 out of 9 for them in everyday life, 90.1% reported the same about the beauty of software architecture. Beauty in the environment is a four or less for 18.1%, while beauty in software architecture is of the same low importance to 9.7% of our participants. Overall, the importance of beauty in software architecture correlates to the beauty in the environment, while the first has slightly higher importance rankings.

Additionally, we analyzed the definition of beauty of the participants that indicated the importance of less than five for the beauty of software architecture. In every answer, we either found similar answers like: "I would not agree that software architecture can be 'beautiful' in a way this term is usually used. Nevertheless, I would totally agree that software can have a high-quality level.", that either denied the concept of beauty in software architecture or had a very technical description of beauty, like: "Clear interfaces, logical structure with no or few redundancies or needless complications.". All of those participants have also marked beauty in their environment to be less important than five.

When we asked developers and software architects how they feel while working on a beautiful software architecture (Q13), over one-third wrote that they feel happy. Positive effects (all codes except "neutral" and "negative emotions") have been described by 78.6%. Clearly, positive emotions have been described by 53.6%. For this, we analyzed the codes

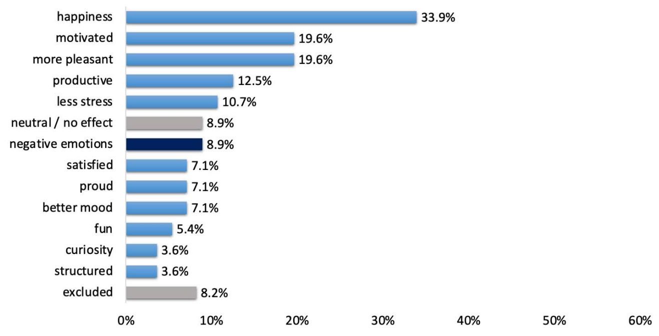

Fig. 5. How developers and architects feel while working on beautiful software architecture.

"curiosity", "fun", "happiness", "motivated", and "proud". Negative emotions were described in 8.9% of the answers, for instance, feeling overwhelmed. In 8.9% of the answers, participants also stated that they do not get affected by the beauty of the software architecture. In more detail, these findings can be inspected in Figure 5.

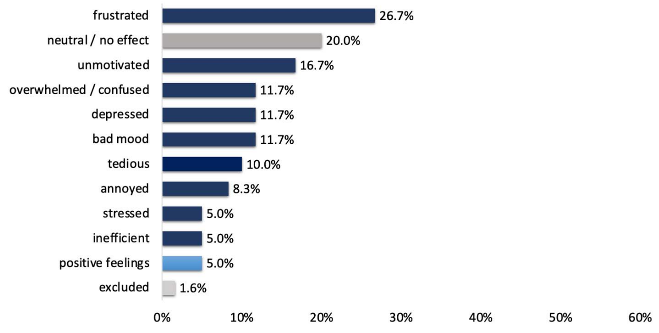

Fig. 6. How developers and architects feel while working on ugly software architecture.

To test the opposite, we have asked developers and software architects how they feel when working on software architecture that they perceive as ugly (Q15). All findings on this question can be examined in detail in Figure 6. Here, 75% stated that they have negative feelings (all codes except for "neutral" and "positive feelings"). The most stated feeling was frustration. Positive feelings were described by 5%. With 20%, the number of people who do not feel affected by ugly software architecture more than doubled compared to the answers (8.9%) of Q13 in Figure 5.

When asked about their motivation, 88.5% of the respondents stated that they feel motivated when working on a project with beautiful software architecture (Q14). When the software architecture of a project is perceived as ugly, 83.6% feel less motivated to work on that project (Q16).

Another question in the survey asked the participants what they would expect of the code of the software with an architecture they perceive as beautiful (Q11). Here, 88.5% stated positive preconceptions like high quality (52.5%), well tested (13.1%), and easy to understand (13.1%). For the positive

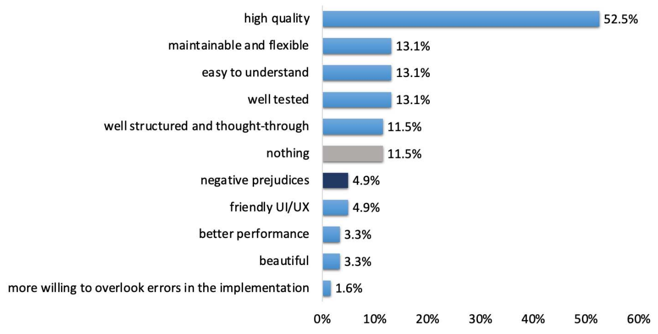

Fig. 7. Preconceptions of the code of a beautiful software architecture.

preconceptions, we included all codes except for "nothing", and "negative preconceptions". Negative preconceptions were mentioned in 4.9% of the answers. Some participants (11.5%) answered that they would expect nothing from the code when they perceive its architecture as beautiful.

When asked in Q12, 44.3% stated that software with beautiful architecture is superior to software without beautiful architecture. This question was the only closed-ended yes/no question that included the option to answer "sometimes".

Key Findings of RQ2
Finding 4: The importance of beauty is at least a 5 of 9 to most participants, while beauty in software architecture got slightly higher rankings than in the environment.
Finding 5: When working on beautiful software architecture, 72.1% of the participants described positive effects, while 73.8% experience negative feelings when working on ugly software architecture.
Finding 6: Beautiful software architecture motivates 88,5% of our participants to work on a project, while ugly software architecture demotivates 83.6%.
Finding 7: Without knowing the code, software is perceived with positive preconceptions like high quality from 88.5% of participants. Overall, 44.3% expect software with beautiful software and architecture to be superior to one without it.

D. RQ3. How can we create beautiful software architecture?

The majority of participants in this study (88%) have already tried to create beautiful software architecture (Q9). When we asked our participants how they try to create one step by step (Q10), the highest ranked answers were: 1) Good requirements engineering (57.4%), 2) Iteration (42.6%), and 3) start with a draft/prototype/MVP (24.6%) / structure appropriately (24.6%).

Reflecting the top three answers and other steps to create beautiful software architecture, we derived a list of seven categories that one must master to be able to create beautiful software architecture, according to the common understanding of beauty among practitioners (RQ1): Beautiful software architecture reduces the effort to work with it.

1) *Requirements engineering:* The system requirements should be understood and clearly defined to enable the design of a beautiful software architecture.
2) *Focus on holistic consistency:* Defining a goal for the system helps to ensure holistic consistency in the design. The "feeling of everything fitting perfectly together" can be achieved by concentrating on holistic consistency. No compromises or workarounds should be allowed in the design of the architecture.
3) *Abstraction:* The system's structure has to be abstracted correctly.
4) *Simplify:* Complex solutions should be visualized and implemented as simply as possible.
5) *Plan for future change:* This helps the system to stay easily extendable and maintainable over time.
6) *Iteration:* Start with a first draft or prototype and iteratively refine it.
7) *Be creative:* Use all your creative capabilities to design beautiful software architecture. Inspiration can be drawn from experience, software patterns, or other sources, if needed.

Most of these categories can be trained, which proposes goals for an effective educational plan.

Key Findings of RQ3
Finding 8: Requirements engineering, focus on holistic consistency, abstraction, simplification, planning for future change, iteration, and creativity help to create beautiful software architecture.

E. RQ4. How can we teach the intuition for beauty in software architecture?

Most findings of RQ3 on how to create beautiful software architecture can be trained. Teaching them is an intuitive answer to the question of how we can teach beauty in software architecture. In the interviews, we discussed in detail with industry experts how to speed up the process of gaining an intuition for its beauty.

The need to train "tools" like patterns and UML modeling has been emphasized by 50% of the experts. UML modeling helps learn how to abstract, simplify and plan for future change, which are three of the mentioned categories in RQ3. These skills are needed to be able to understand and create beautiful software architecture. Additionally, 75% of the experts stressed that working with stunning and ugly software architecture is needed to gain the most formative experiences.

From the interviews, we have derived two ways to teach beauty in software architecture and developed one educational plan that can easily be implemented in the industry.

1) *Teach the methods:* Many different methods that are needed to identify and create beautiful software architecture can be taught. These include software architectural patterns, UML modeling, and many other tools.
2) *Work in extremes:* Beauty in software architecture has to be experienced. Extreme examples help shape the

personal understanding of beauty in software architecture the most. This means that working on projects with magnificent software architecture and ones with a hideous one, for instance, will help to make the most important experiences as early as possible. Ugly software architecture can often be found in projects with critical time and budget constraints. This will help to value beautiful software architecture more and to be able to shape one's perception of beauty in software architecture.

3) *The Buddy-Program:* In every new project, an experienced software architect has to design the new system. The people undertaking the buddy program now have this expert as their buddy for two to three days. For the first third of the program, the junior will get the same task as the expert and will try to design the system themselves. Afterward, the junior will meet with their buddy, and they will discuss and compare both of their designs in detail. This way, the junior can learn a lot from their buddy and from their own mistakes that they have made during their hands-on design experience. Additionally, the design of the original architecture might be improved with the ideas of the junior.

Key Findings of RQ4
Finding 9: Different methods, like patterns and UML modeling can be taught. Working in extremes, with very beautiful and very ugly software architecture, and the buddy program can help to make the most formative experiences as early as possible.

V. Discussion

A. Discrepancies between the Definition of Beautiful Software Architecture in Literature and Practice

Beautiful software architecture can be defined by its "commercial success, influence on other product line architectures (others have "borrowed, copied, or stolen" from the architecture), and sufficient documentation" [2] or by having almost perfect symmetries [3] in literature.

Most practitioners define software architecture as something that cannot be seen, according to our question about the definition of software architecture in the survey (Q3). This indicates that practitioners think that diagrams are only used to describe the architecture and are not part of it, which is why only 6.6% perceive the software architecture as beautiful when it is visually beautiful. Separating a software system's abstract, transparent architecture from its diagrams is a very logical approach to software architecture, as diagrams can be changed, and multiple diagrams can describe the same software architecture from different views. Architecture descriptions should not determine its quality, adequacy, or beauty, and our data indicate that practitioners make this separation correctly.

In contrast to the practitioners, the literature does include descriptions of software architecture in its definition. These discrepancies can appear since defining software architecture is complicated because "software development does not have a unique architectural level of design" [47]. Additionally, it is incredibly challenging to separate the system's software design and architecture from all its details. [48].

As we found that, in 88.5% of the responses, the participants stated that beautiful software architecture makes software easier to work with (Finding 1), and almost two-thirds of answers indicated that beautiful software architecture reduces complexity (Finding 2), we think that there seems to be a link. Overall, the common understanding among practitioners is that beautiful software architecture reduces their effort to work with it. Therefore we conclude that software architecture that makes the job of software developers and architects easier and faster is perceived as beautiful by them.

This is a very functional definition of beauty. A similar one can be found in civil architecture as well. In functionalism, beauty comes from the fulfillment of the function for which the building was constructed [16], [17]. Beauty and usefulness of a building are also both equally important for Vitruvius, which emphasizes that one cannot create beautiful buildings that are useless [18]. Civil and software architecture have similar functional definitions for beauty, as they both have some intended functionality in every design.

B. The Importance of Beauty in Software Architecture

Looking at the common understanding of beauty in software architecture as reducing the effort to work with it, beauty in software architecture is beneficial in every project. This is also a cost and time-reducing argument to advocating beauty in software architecture. Most practitioners already recognize the importance of beauty, as 90.1% ranked the importance of beauty in software architecture at least a 5 out of 9. The ones that did not find beauty essential in software architecture either denied that beauty exists in software architecture or had a very technical description of beauty. Therefore, these answers do not oppose the statement that beauty in software architecture is important.

Software architecture also influences the emotions of the developers. When working with beautiful software architecture, 78.6% of the participants perceive positive effects, while 75% perceive negative feelings while working on a project with ugly software architecture (Finding 5). According to these results, working with ugly software architecture can potentially negatively influence the mental health of its developers. This has a negative impact on the developers' motivation and, as a result, the system's cost and quality over time.

Instead, the software architecture should delight its developers and testers. Satisfaction makes the developers' jobs easier and is more likely to result in a high-quality system. This is analogous for testers [2]. Beauty in software architecture can, therefore, directly lead to higher-quality systems. This is relatable, because when beauty in software architecture means understanding a software architecture easier and faster and integrating changes very quickly, the software developers can enjoy their work more. This is also reflected in the many positive effects of working with beautiful software architecture (Finding 5). Developers can be more focused on integrating functionality, without being distracted by solving problems and trying to understand a chaotic structure. Therefore, it is necessary to prioritize beauty in the software architecture of mid- to large-sized projects to improve the happiness and motivation of its developers and software architects, as well as the productivity of the development team.

Immanuel Kant strictly separates beauty and functionality by defining beauty as what delights without looking at its purpose. This definition excludes the practical use of beauty, which could raise the question of the practical relevance of non-functional beauty in software architecture. As Kant does not exclude the delight of beauty with his definition, we can find the relevance of beauty in software architecture in his definition as well. Delighted developers produce higher quality systems [2]. However, as beauty in software architecture is not defined non-functional, but functional, just like in civil architecture, these thoughts are only theoretical [16], [17], [18].

"We know from experience that we should evaluate an architecture to determine whether it will meet its requirements before spending money to build, test, and deploy the system" [2]. Beauty ought to be considered when evaluating architecture due to its impact on the developers and, consequently, the project's overall success. Not only does beauty in software architecture influence the developers while working on it, but it also influences their assumptions about a software system. Without knowing the code, the software is perceived with positive preconceptions like high quality by 88.5% of participants. Overall, 44.3% perceive software with beautiful software architecture as superior to one without (Finding 7). When customers only have a few hours to examine a software system before accepting it, the architecture's beauty significantly impacts their perception of the system and its quality. This means that all software projects should focus on the beauty of their software architecture before shipping it to customers that keep on working on the code. When discussing preconceptions, it is crucial to what one looks at to evaluate the beauty of architecture. It could be static or dynamic diagrams of a system, or many other possibilities to perceive an architecture. The perspective on the architecture did not play a role in evaluating our question about preconceptions because we asked the participants what they thought of a system if they perceived its architecture as beautiful and had not seen the code. What helped them to conclude that the software architecture is beautiful was not targeted.

Overall, beauty in software architecture is essential in any project, especially in mid- to large-sized projects. Therefore, research and the education of software developers and architects should increase in that field.

C. *The Creation of Beautiful Software Architecture*

Beauty in software architecture can be created with reasonable requirements engineering, a focus on holistic consistency, abstraction, simplification, planning for future change, iteration, and creativity (Finding 8). Simplifying is one of the most challenging aspects of these. Breaking down a complex solution and making it appear simple and easy to understand requires the person that simplifies to be an expert in the topic. In a work environment, simple solutions are hardly valued for the time and effort that have been put into making the solution seem simple. Furthermore, many practitioners like to be admired by their peers for understanding very complex solutions. Simplifying it might lead to losing admiration, which decreases their motivation for simplification in the first place. The laws of simplicity are very well described in Maeda's book with the same name [65].

The categories to create beautiful software architecture correlate with the answers to the question about beautiful software architecture. Most of the categories help to create an architecture that is easy to understand, extend, and maintain.

An iterative approach is sometimes not helpful when working with legacy systems, for instance, as it is sometimes needed to redesign architecture to make it beautiful entirely. Additionally, one has to keep in mind that after a certain amount of iterations without significant beautification, one should stop and start again. This can be very hard sometimes, especially when much work has already been put into the project. However, knowing when to stop and start over is essential to create beautiful software architecture.

D. *The Education of Beauty in Software Architecture*

What is perceived as beautiful can be subjective, but in our study, we have found a common understanding of practitioners. Most practitioners perceive software as beautiful if it reduces their effort to work with it.

A combination of active (methods) and passive (experience) education can help speed up developing one's sense of beauty in software architecture. This re-emphasizes that beauty has to be experienced and is not understood only by active education, which highlights the link between beauty and utility in this context. Developers have to experience how hard it can be to create something beautiful and how frustrating it might be to work on an ugly architecture to appreciate the beauty in other architectures truly. Additionally, developers have to experience various magnificent architectures to get inspired.

For the buddy program, one needs an expert architect as a buddy. Often senior software architects are scarce, leading to them consulting multiple projects at a time. This is not a problem for the buddy-program approach because striving junior architects can work on the problem statement of another project than their own. The respective buddy from their buddy-program project does not have to have created the whole architecture themselves but only needs to be an expert in architecture and should know the problem domain and the architecture design of the project well enough to be able to assess the design of the program participant constructively. Of course, one has to ensure that this experience is positive overall for both the buddy and the program participant.

E. *Threads to Validity and Limitations*

As we have a limited amount of 61 valid datasets, the external validity of our statistics can be increased with more study participants. Whether the 61 participants in this study are a representative sample of software developers and architects poses a pertinent risk. We tried to alleviate this threat by sending invitations to participants from various sources. Additionally, we applied the snowball sampling method [60], which helped to reach a broader audience and therefore increase the heterogeneity of our participants. Furthermore, the responses show a heterogenous sample in terms of professional experience. This raises confidence that the study's findings and results do not suffer from severe biases concerning the methods used to invite participants and indicates a good spread of participant profiles.

We are aware that our sampling could entail a selection bias. Some developers and architects might have not participated in our study because they were not interested in the topic. Additionally, the ones that participated might already have a greater interest in the beauty of software architecture. However, our answers entail various opinions on beautiful software architecture. Additionally, we have utilized our industry contact to interview some industry experts. Their insights have been used in RQ4 and could entail some bias, as they are all from the same company and, therefore, most likely have similar training and a similar mindset in the first place.

When examining developers' emotions while working with beautiful and ugly software architecture (RQ2), it might be possible that the answers were mixing working with software architecture and code. We tried to minimize this risk by asking preceding questions about their definition of software architecture and its beauty. This helped the developers think about the concept of software architecture independent from the code of a system which should help them answer the questions about their emotions in the way we intended.

The qualitative analysis of RQ3 and RQ4 could threaten the validity of our conclusion. Also, our education guidelines have yet to be evaluated. The correctness of these statements still needs to be proven. All findings have been discussed with industry experts, which helps to mitigate these threats.

Respondent fatigue bias has been dealt with by limiting the number of questions to 16. Only posing five open-ended questions helped to reduce the time needed to complete this study even further. The whole survey should have been able to complete in less than 15 minutes, which was tested in the pilot study as well.

Furthermore, manually coding the open-ended questions could have introduced some personal bias. Being aware of that from the beginning, the authors helped each other to reduce this bias by discussing all difficulties.

The correctness and comprehensibility of our survey and interview questions are another thread of validity. By piloting our study and interview, we have iterated over the survey and interview questions to alleviate this risk. For the closed-ended Yes/No questions, some participants might not have found any suitable answers. After analyzing the potential bias in these types of questions, the option "sometimes" was introduced for the questions we evaluated as being at risk.

We address reliability and replication concerns by providing all our interview and survey questions and datasets upon request. This makes it possible for different researchers to assess the rigor of the design and replicate the study.

F. Future Research

While working on software architecture's beauty, we also came up with a number of interesting theories that can be studied in the future. It may be sufficient to perceive software architecture as beautiful on a high level to perceive it as beautiful overall. This means that on a macro level, beauty is essential, while it is not so important on a micro level anymore. This would save time in the industry because beautification efforts could focus on the macro-architectural level without compromising the overall perception of architecture's beauty. Additionally, symmetry and its repetition are perceived as beautiful by many individuals [1]. If this applies to software architecture as well, fractal-like structures in software architecture could help increase its beauty. Another interesting investigation would be whether an architecture could be too perfect, as some participants have negative feelings while working with beautiful software architecture, like feeling overwhelmed. Possibly, a software architecture needs some flaws to be more beautiful. Also, cultural differences play an essential role in the perception of beauty. They could therefore be found in the perception of beauty in software architecture as well, upon further investigation [1], [30], [31], [36].

VI. Conclusion

According to common understanding among practitioners, beautiful software architecture reduces the effort to work with it. Beauty delights and motivates developers and therefore increases the productivity of a team and the quality of the product. Conversely, working with ugly software architecture frustrates and demotivates developers, decreasing productivity and product quality. When working on mid- to large-sized projects, these effects can amplify, highlighting the importance of beautiful software architecture in those projects. To create beautiful software architecture, one has to focus on reasonable requirements engineering, holistic consistency, abstraction, simplification, planning for future change, iteration, and creativity. Several of these topics can be taught, which proposes potential topics for an educational plan. Even though a lot of methods, like UML modeling or software patterns can be taught and can increase the capability to create and perceive beauty in software architecture, it has to be experienced in practice. Therefore working with magnificent and hideous software architecture can help to make the essential experiences earlier. Additionally, we have proposed the buddy program to enable hands-on architectural experience in an industrial setting.

Acknowledgments

The authors acknowledge the insightful comments on our research from Jonas Schulte-Coerne and Markus Schnappinger and would like to thank them for their valuable input.

References

[1] A. Sisti, N. Aryan, and P. Sadeghi, "What is beauty?" *Aesthetic Plastic Surgery*, vol. 45, no. 5, pp. 2163–2176, 2021.

[2] D. Spinellis and G. Gousios, *Beautiful architecture: leading thinkers reveal the hidden beauty in software design*. O'Reilly Media, Inc., 2009.

[3] B. Appleton, "Patterns and software: Essential concepts and terminology," 1997.

[4] T. Fujino, "Traditional japanese architecture blends beauty and rationale [lessons for software engineering]," *IEEE software*, vol. 16, no. 6, pp. 101–103, 1999.

[5] I. Kant, K. Vorländer, and H. Klemme, *Kritik der urteilskraft*. F. Meiner, 1924, vol. 1.

[6] P. N. Broer, S. Juran, M. E. Walker, R. Ng, K. Weichman, N. Tanna, Y.-J. Liu, A. Shah, A. Patel, J. A. Persing *et al.*, "Aesthetic breast shape preferences among plastic surgeons," *Annals of Plastic Surgery*, vol. 74, no. 6, pp. 639–644, 2015.

[7] S. Zeki, "Beauty in architecture: Not a luxury-only a necessity," *Architectural Design*, vol. 89, no. 5, pp. 14–19, 2019.

[8] M. Agius, "What is beauty? should doctors point out beauty to their patients during therapy?" *Psychiatria Danubina*, vol. 30, no. suppl. 7, pp. 555–558, 2018.

[9] S. H. Dayan, R. T. Cristel, N. D. Gandhi, S. G. Fabi, O. J. Placik, J. R. Montes, and A. Kalbag, "Perception of beauty in the visually blind: A pilot observational study," *Dermatologic Surgery*, vol. 46, no. 10, pp. 1317–1322, 2020.

[10] D. S. Hamermesh and J. Abrevaya, "Beauty is the promise of happiness?" *European Economic Review*, vol. 64, pp. 351–368, 2013.

[11] K. Dion, E. Berscheid, and E. Walster, "What is beautiful is good." *Journal of personality and social psychology*, vol. 24, no. 3, p. 285, 1972.

[12] A. Feingold, "Good-looking people are not what we think." *Psychological bulletin*, vol. 111, no. 2, p. 304, 1992.

[13] M. E. Van den Elzen, S. L. Versnel, S. E. Hovius, J. Passchier, H. J. Duivenvoorden, and I. M. Mathijssen, "Adults with congenital or acquired facial disfigurement: impact of appearance on social functioning," *Journal of Cranio-Maxillofacial Surgery*, vol. 40, no. 8, pp. 777–782, 2012.

[14] T. Wang, L. Mo, C. Mo, L. H. Tan, J. S. Cant, L. Zhong, and G. Cupchik, "Is moral beauty different from facial beauty? evidence from an fmri study," *Social cognitive and affective neuroscience*, vol. 10, no. 6, pp. 814–823, 2015.

[15] S. Stephens, *Tenacious beauty: The shifting role of aesthetic criteria in architecture criticism, 1850–1915*. Cornell University, 2002.

[16] E. R. DeZurko, "Greenough's theory of beauty in architecture," *Rice Institute Pamphlet-Rice University Studies*, vol. 39, no. 3, 1952.

[17] R. Alasmar, "Philosophy and perception of beauty in architecture," *American Journal of Civil Engineering*, vol. 7, no. 5, pp. 126–132, 2019.

[18] M. P. Vitruvius, *De architectura libri decem*. Teubneri, 1867.

[19] C. Alexander, *A pattern language: towns, buildings, construction*. Oxford university press, 1977.

[20] D. Lea, "Christopher alexander: An introduction for object-oriented designers," *ACM SIGSOFT Software Engineering Notes*, vol. 19, no. 1, pp. 39–46, 1994.

[21] C. Alexander, *Notes on the Synthesis of Form*. Harvard University Press, 1964, vol. 5.

[22] C. Alexander *et al.*, *The timeless way of building*. New york: Oxford university press, 1979, vol. 1.

[23] A. A. Brielmann and D. G. Pelli, "Beauty requires thought," *Current Biology*, vol. 27, no. 10, pp. 1506–1513, 2017.

[24] C. W. Valentine, *The experimental psychology of beauty*. Routledge, 2015.

[25] P. Sarasso, I. Ronga, P. Kobau, T. Bosso, I. Artusio, R. Ricci, and M. Neppi-Modona, "Beauty in mind: Aesthetic appreciation correlates with perceptual facilitation and attentional amplification," *Neuropsychologia*, vol. 136, p. 107282, 2020.

[26] K. Grammer and R. Thornhill, "Human (homo sapiens) facial attractiveness and sexual selection: the role of symmetry and averageness." *Journal of comparative psychology*, vol. 108, no. 3, p. 233, 1994.

[27] S. W. Gangestad, R. Thornhill, and R. A. Yeo, "Facial attractiveness, developmental stability, and fluctuating asymmetry," *Ethology and Sociobiology*, vol. 15, no. 2, pp. 73–85, 1994.

[28] D. B. Sarwer, T. A. Grossbart, and E. R. Didie, "Beauty and society." in *Seminars in Cutaneous Medicine and Surgery*, vol. 22, no. 2, 2003, pp. 79–92.

[29] F. Vegter and J. J. Hage, "Clinical anthropometry and canons of the face in historical perspective," *Plastic and reconstructive surgery*, vol. 106, no. 5, pp. 1090–1096, 2000.

[30] P. W. Hashim, J. K. Nia, M. Taliercio, and G. Goldenberg, "Ideals of facial beauty." *Cutis*, vol. 100, no. 4, pp. 222–224, 2017.

[31] J. Milutinovic, K. Zelic, and N. Nedeljkovic, "Evaluation of facial beauty using anthropometric proportions," *The Scientific World Journal*, vol. 2014, 2014.

[32] H. E. Huntley, *The divine proportion*. Courier Corporation, 2012.

[33] A. Sisti and P. Sadeghi, "Leonardo da vinci: the lady with an ermine enigma," *Hormones*, vol. 20, no. 3, pp. 591–592, 2021.

[34] T. Greywal, S. H. Dayan, K. Goldie, and S. Guillen Fabi, "The perception bias of aesthetic providers," *Journal of Cosmetic Dermatology*, vol. 20, no. 6, pp. 1618–1621, 2021.

[35] P. N. Broer, S. Juran, Y.-J. Liu, K. Weichman, N. Tanna, M. E. Walker, R. Ng, and J. A. Persing, "The impact of geographic, ethnic, and demographic dynamics on the perception of beauty," *Journal of Craniofacial Surgery*, vol. 25, no. 2, pp. e157–e161, 2014.

[36] P. Sadeghi and A. Sisti, "Forehead fat grafting: Asian facial contouring and augmentation," *Plastic and Reconstructive Surgery*, vol. 146, no. 4, pp. 499e–500e, 2020.

[37] W. Dawei, Q. Guozheng, Z. Mingli, and L. G. Farkas, "Differences in horizontal, neoclassical facial canons in chinese (han) and north american caucasian populations," *Aesthetic plastic surgery*, vol. 21, no. 4, pp. 265–269, 1997.

[38] L. Farkas, C. Forrest, and L. Litsas, "Revision of neoclassical facial canons in young adult afro-americans," *Aesthetic Plastic Surgery*, vol. 24, no. 3, pp. 179–184, 2000.

[39] M. R. Cunningham, A. R. Roberts, A. P. Barbee, P. B. Druen, and C.-H. Wu, ""their ideas of beauty are, on the whole, the same as ours": Consistency and variability in the cross-cultural perception of female physical attractiveness." *Journal of personality and social psychology*, vol. 68, no. 2, p. 261, 1995.

[40] A. Furnham and P. Baguma, "Cross-cultural differences in the evaluation of male and female body shapes," *International Journal of Eating Disorders*, vol. 15, no. 1, pp. 81–89, 1994.

[41] S. Rhee, "Differences between caucasian and asian attractive faces," *Skin Research and Technology*, vol. 24, no. 1, pp. 73–79, 2018.

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

[43] R. Li, M. Soliman, P. Liang, and P. Avgeriou, "Symptoms of architecture erosion in code reviews: a study of two openstack projects," in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*. IEEE, 2022, pp. 24–35.

[44] G. Booch, "On architecture," *IEEE software*, vol. 23, no. 2, pp. 16–18, 2006.

[45] B. Meyer, "Software architecture: Object-oriented vs functional," *i Beautiful Architecture, Sebastopol, O'Reilly Media*, pp. 331–346, 2009.

[46] ——, *Object-oriented software construction*. Prentice hall Englewood Cliffs, 1997.

[47] J. Baragry and K. Reed, "Why is it so hard to define software architecture?" in *Proceedings 1998 Asia Pacific Software Engineering Conference (Cat. No. 98EX240)*. IEEE, 1998, pp. 28–36.

[48] 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.

[49] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to advanced empirical software engineering*. Springer, 2008, pp. 63–92.

[50] M. J. de Dieu, P. Liang, and M. Shahin, "How do developers search for architectural information? an industrial survey," in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*. IEEE, 2022, pp. 58–68.

[51] M. Freyd, "The graphic rating scale." *Journal of educational psychology*, vol. 14, no. 2, p. 83, 1923.

[52] P. De Jong, J. M. E. Van Der Werf, M. Van Steenbergen, F. Bex, and M. Brinkhuis, "Evaluating design rationale in architecture," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2019, pp. 145–152.

[53] G. Vale, F. F. Correia, E. M. Guerra, T. de Oliveira Rosa, J. Fritzsch, and J. Bogner, “Designing microservice systems using patterns: An empirical study on quality trade-offs,” in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*. IEEE, 2022, pp. 69–79.
[54] S. E. Hove and B. Anda, “Experiences from conducting semi-structured interviews in empirical software engineering research,” in *11th IEEE International Software Metrics Symposium (METRICS'05)*. IEEE, 2005, pp. 10–pp.
[55] C. B. Seaman, “Qualitative methods,” in *Guide to advanced empirical software engineering*. Springer, 2008, pp. 35–62.
[56] P. Ralph, N. b. Ali, S. Baltes, D. Bianculli, J. Diaz, Y. Dittrich, N. Ernst, M. Felderer, R. Feldt, A. Filieri *et al.*, “Empirical standards for software engineering research,” *arXiv preprint arXiv:2010.03525*, 2020.
[57] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, “Design, monitoring, and testing of microservices systems: The practitioners' perspective,” *Journal of Systems and Software*, vol. 182, p. 111061, 2021.
[58] F. Tian, P. Liang, and M. A. Babar, “Relationships between software architecture and source code in practice: An exploratory survey and interview,” *Information and Software Technology*, vol. 141, p. 106705, 2022.
[59] G. Uddin, O. Baysal, L. Guerrouj, and F. Khomh, “Understanding how and why developers seek and analyze api-related opinions,” *IEEE Transactions on Software Engineering*, vol. 47, no. 4, pp. 694–735, 2019.
[60] B. A. Kitchenham and S. L. Pfleeger, “Principles of survey research part 2: designing a survey,” *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 1, pp. 18–20, 2002.
[61] K.-J. Stol, P. Ralph, and B. Fitzgerald, “Grounded theory in software engineering research: a critical review and guidelines,” in *Proceedings of the 38th International conference on software engineering*, 2016, pp. 120–131.
[62] C. Wohlin, M. Höst, and K. Henningsson, “Empirical research methods in software engineering,” in *Empirical methods and studies in software engineering*. Springer, 2003, pp. 7–23.
[63] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.
[64] “ISO/IEC/IEEE 42010:2022 - Software, systems and enterprise - Architecture description,” International Organization for Standardization, Geneva, CH, Standard, Nov. 2022.
[65] J. Maeda, *The laws of simplicity*. MIT press, 2006.