Bringing Distributed Collaborative Design and Team Collaboration to the Table: A Conceptual Framework

Mahum Adil[a], Ilenia Fronza[b] and Claus Pahl[c]
Free University of Bozen-Bolzano, Italy

Keywords: Architecture Design Decision (ADD), Distributed Software Development, Global Software Engineering (GSE), Distributed Collaborative Design (DCD), Scrum.

Abstract: *Background.* The recent rise of software organizations shifting towards distributed environments increased the feed for distributed collaborative design (DCD), which requires dedicated design thinking and decision strategies to provide a clear architecture plan for distributed teams. *Aim.* This study aims to provide effective support for distributed development teams to enhance the transparency of architecture design decisions and contribute towards clear documentation of the decision rationales. *Method.* Using an action research model, we propose a conceptual framework for distributed Scrum software development teams to manage distributed collaborative design. *Results.* The proposed framework consists of two phases in which the team performs activities to decrease collaboration barriers in design thinking and decision-making. *Conclusions.* The illustrating example concretely shows the proposed conceptual framework's potential and effectiveness in supporting DCD for team collaboration. Further empirical evaluation and validation of the framework are needed in real-time environments.

1 INTRODUCTION

Software design aims at discovering architectural elements and artifacts while making design decisions and defining relationships among system components (Qin et al., 2008). The success of a software system directly depends on this activity, which supports efficient maintenance, long-term use, and continuous evolution in a changing environment (Kruchten et al., 2006). Over the years, the representation of software design evolved from an artifact to a set of Architecture Design Decisions (ADD) (Venters et al., 2018).

Software architecture design represents one of the biggest challenges for distributed teams (Sievi-Korte et al., 2019). There is limited research on *Distributed Collaborative Design* (DCD), which is impacted by distance across several aspects, such as cognition and communication (Sawyer and DeZutter, 2009), understanding, evaluation, and negotiation of concepts (Rice et al., 2007), and the number of discussions about the design problems (Eris et al., 2014). These issues need to be addressed to build a shared understanding of the requirements, record ADDs, and collectively decide the final set of ADDs for the software system (Yang et al., 2021). With the rise of software organisations shifting towards distributed development (Smite et al., 2021), research focused on supporting the ADD process (Parizi et al., 2022; Razavian et al., 2019). However, the factors influencing decision-making need to be addressed.

This work proposes a framework to *support Agile (Scrum) software development teams during DCD to produce the ADDs of a software system*. The focus on Agile is motivated by its widespread adoption, with substantial growth in 2021 (Digital AI, 2021) to address most of the complications related to the distributed environment (Fronza et al., 2022; Vallon et al., 2018; Adil et al., 2022). Therefore, software education and organizations increasingly need guidelines to implement team collaboration in agile distributed teams (Cico et al., 2021). Thus, the proposed solution will be beneficial both in software engineering education and in supporting professional teams.

The paper is structured as follows. Section 2 describes the exiting literature, Section 3 presents the research objective, and Section 4 introduces the proposed conceptual framework. Section 5 details a sample application, and Section 6 concludes the paper and discusses the future work for the study.

[a] https://orcid.org/0000-0001-6452-6085
[b] https://orcid.org/0000-0003-0224-2452
[c] https://orcid.org/0000-0002-9049-212X

Adil, M., Fronza, I. and Pahl, C.
Bringing Distributed Collaborative Design and Team Collaboration to the Table: A Conceptual Framework.
DOI: 10.5220/0011820500003467
In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023) - Volume 2*, pages 126-133
ISBN: 978-989-758-648-4; ISSN: 2184-4992

2 RELATED WORK

Software architecture forms the basis for software architects to distinguish significant architectural design decisions (Jansen and Bosch, 2005), which capture architectural elements and constraints to determine the development and evolution of the software system (Babar et al., 2009). Bosch highlighted the importance of describing software architecture as a set of ADDs (Bosch, 2004), where each ADD is the outcome of the software design process, which comprises software elements that evolved during the software development process (Alexeeva et al., 2016).

Several approaches have been proposed to visualize software architecture as explicit design decisions (Kruchten et al., 2009), capture and visualize architectural elements (e.g., (Jansen and Bosch, 2005; Capilla et al., 2006), evaluate quality attributes (e.g., (Babar and Capilla, 2008; Lytra et al., 2013)), automatically extract ADD and support *collaboration* to evaluate alternatives (e.g., (Nowak and Pautasso, 2013; Hesse et al., 2016)). Existing research on ADD, however, did not explore the factors influencing decision-making (Parizi et al., 2022; Razavian et al., 2019). *Collaborative design* requires intensive design thinking and decision strategies to provide a clear architecture plan for distributed teams (Tofan et al., 2014): 86% of ADDs in software organisations are based on group decisions to improve software design quality (Tofan et al., 2013). The team members rely on software design knowledge, productive communication, and problem-solving skills to enhance the design decision process.

Distributed collaborative design (DCD) faces several challenges: collaboration is impacted by distance across several aspects, such as cognition and communication (Sawyer and DeZutter, 2009), understanding, evaluation, and negotiation of concepts (Rice et al., 2007), and several discussions about the design problems (Eris et al., 2014). Several computer-aided systems exist to support (synchronous or asynchronous) communication and interaction between collaborators in distributed environments (Safin et al., 2012). Instead, few studies explored how the design activities involving information–problem–solution (i.e., IPS design activities) differ between co-located and distributed environments. Existing research shows that distance has a minor effect on the problem exploration to design and structure decision activities (Yang et al., 2021), and various approaches exist to manage architectural knowledge for software development (Borrego et al., 2017; Portillo-Rodríguez et al., 2012). However, collaboration practices in a distributed environment are not explored. Moreover, limited research on collaborative design decisions (Muccini and Rekha, 2018) highlights a current gap in architecture knowledge formation practices to manage ADDs in distributed teams.

3 RESEARCH OBJECTIVE AND METHODOLOGY

This study aims at supporting Agile (Scrum) software development teams during DCD to produce the ADDs of the software system. Based on this goal, we derived the following research question:

RQ: *How can we effectively support distributed collaborative design to review the architecture design decisions of the system?*

The research methodology of this study is based on the action research model, which focuses on improving and keeping a balance of current software engineering practices with regard to the software industry (Kemmis and McTaggart, 2007). The model consists of four steps in the iteration:

1. Plan: planning in order to initiate change. This step has been completed by analyzing existing literature and practices to define the requirements of the *change* (Section 4.1).
2. Act: implementing the change. This step has been completed by designing a conceptual framework (Section 4.2).
3. Observe: observe the consequences of the change. The focus of this work is on the first two steps of the action research model; the first part of the *observe* step (i.e., a demonstrating example) is described in Section 5.
4. Reflect: reflect on the results and re-plan.

4 PROPOSED SOLUTION

This section illustrates our proposed solution: a conceptual framework for distributed Scrum software development teams to manage DCD.

4.1 Requirements and Main Characteristics

Requirements have been collected in the *plan phase* of the action research model (Section 3) by analyzing the existing literature and by informal interviews with experts in the field and practitioners. The rest of this section describes the characteristics of the proposed framework mapped to each requirement (Table 1).

Table 1: Mapping between requirements and characteristics of the framework.

Requirement	Characteristics
Support Scrum teams	Scrum elements such as team roles, iterations and activities
Increase the lifespan of ADDs	Design forces (Van Heesch et al., 2012) to elicit functional or non-functional factors that can impact ADR
Increase sustainability of decisions and reduce efforts for documentation	Y-statement model (Zdun et al., 2013) for Architectural Decision Records (ADR) documentation
Improve traceability of design decisions and emphasize the importance of team consensus	Evidence Criteria - Architectural Decision Records (EC-ADR) checklist (Zimmermann, 2021) to evaluate design forces with ADR
Support distributed software development	The Y-statement model and the EC-ADR checklist are integrated in a framework to support DCD

Support Scrum Teams. The paradigm shift in the workplace environment fueled by COVID-19 has resulted in the rise of studies investigating the transition from working in a co-located environment to distributed environment (Ralph et al., 2020). Scrum methodology is becoming mainstream in software organizations to increase collaboration between distributed teams (Lous et al., 2017). To tailor the proposed framework to Scrum teams, we characterized team roles, iterations, and activities as part of the transition into an agile environment.

Increase the Lifespan of the ADDs. Eliciting and documenting design rationales for architecture decisions is crucial for software development systems. *Decision forces* make design decisions transparent and document functional or non-functional factors that can potentially impact the quality of a software system (Van Heesch et al., 2012). Based on the existing research (Weinreich et al., 2015; Rueckert et al., 2019), we described the design forces into six categories. These forces will help establish traceability between design decisions and the factors (functional or non-functional) that influence the decision.

Increase Sustainability of Decisions and Reduce Documentation Effort. The framework includes the *Y-statement model* (Zdun et al., 2013), a lightweight decision record template to document task-specific decisions to formulate ADDs. This model has been selected based on a comparative analysis presented that found this model helpful in capturing the structure and elements of design attributes to build a design decision storyline (Zimmermann et al., 2015).

Improve Traceability of Design Decisions and Emphasize Team Consensus. In order to bring the quality of conformance in design, there are various software quality assessment practices that focus on the evaluation of non-functional requirements (Lytra et al., 2020). However, non-functional requirements have been neglected so far. With the goal of evaluating functional and non-functional requirements from the architecture record, the proposed framework includes the *Evidence Criteria - Architectural Decision Records (EC-ADR)*, i.e., a checklist to analyze whether a design decision provides enough rationale for the decision outcome (Zimmermann, 2021). The EC-ADR checklist (Figure 1) is based on five elements named evidence, criteria, agreement, document, and review to promote group decision-making practice.

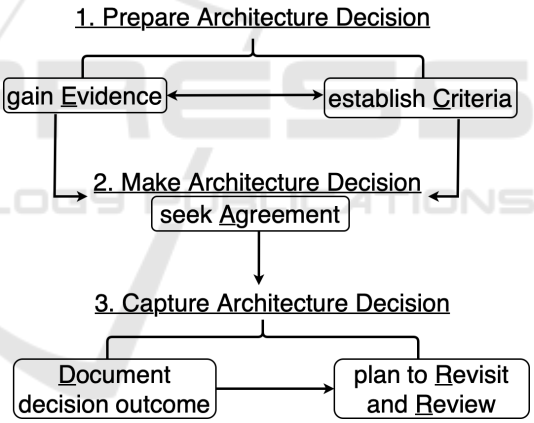

Figure 1: EC-ADR checklist to evaluate architecture decision (Zimmermann, 2021).

Support Distributed Software Development. In literature, the *Y-statement* model and the *EC-ADR checklist* are used to help co-located software architects construct the design rationale for the ADD process. To support DCD, i.e., to enhance the ADD process through collaborative design practices (Muccini and Rekha, 2018), we integrated these methods in the proposed framework, which can be executed by distributed software development teams using Scrum methodology to foster group decisions and collaboration.

4.2 Structure of the Conceptual Framework

The proposed framework supports synchronous DCD activities, i.e., distributed teams will follow a reverse engineering approach (Chikofsky and Cross, 1990) to build and evaluate ADD records and to decrease the collaboration barriers for participating in the design thinking and decision-making process.

As shown in Figure 2, the framework is divided into two phases. In the *pre-sprint event*, the product owner coordinates with the scrum master to identify architecture design attributes to build the design rationale of the software system. In the *sprint event*, the development team (i.e., scrum master, design analyst, and lead developer) performs the following collaborative activities: (i) thorough elaboration on the significance of design attributes and their impact on the software system, (ii) production of lean documentation stating design decisions, by listing all related design attributes with the possible set of solution, and (iii) analysis and group discussion to evaluate the feasibility of the ADDs.

4.2.1 First Phase: Pre-Sprint Event

Step 1: Identify Architecture Design Attributes. The scrum master and the product owner use a set of requirements to identify stakeholders, design objectives, and architecture design attributes that result in new architecture design metrics or existing ADDs. Both new and existing requirements are considered input data: existing ADDs consist of design decisions extracted from an existing system to influence the design, while architecture design metrics are extracted from new requirements to identify all the relevant information (e.g., domain assumption, use case definition with respect to actors, and architecture drivers). The output of this phase is a description of all architecture design elements, which will be used by the scrum master to build the design rationale with the development team.

4.2.2 Second Phase: Sprint Event

Step 2: Architecture Design Significance. In order to avoid any architecture knowledge vaporization (Bosch, 2004), the proposed framework requires the development team to do group discussions to determine the architecture design significance by looking at existing ADDs or architecture design metrics describing new ADDs. If the architecture significance is determined, the related design elements become part of the DCD decision. Otherwise, they will be implemented directly by the core development team.

Step 3: Classify Design Forces. To elicit and document the list of ADDs during the design process, the distributed development team will do group discussions to build a preliminary list of design forces and classify them into the following six categories (Rueckert et al., 2019):

1. Functional: main functionalities of the software system (e.g., concurrency handling, network communication, and data storage).
2. Non-functional: quality attributes that define the behavior of the system.
3. Third-party technology: specific to knowledge, expected license cost, and support of commercially available open-source libraries or middleware.
4. Design principles and guidelines: fundamental principles, such as guidelines for the naming convention and design framework usage.
5. Technology selection/implementation: related to the selection of technology that concerns the competitiveness of the software system and team knowledge and experience in the programming language.
6. Deployment: time to conduct testing and to share a prototype.

Step 4: Create Architecture Design Records (ADRs). In this step, the design analyst and the software lead developer use the Y-statement model (Zdun et al., 2013) to form lean documentation of ADRs by presenting ADDs with related design forces based on the following template:

> *In the context of* <use case/user story **u**>, *facing* <concern **c**>, *we decided for* <option **o**>, *and neglected* <other options>, *to achieve* <system qualities, desired consequence **q**>, *accepting* <downside/undesired consequence **d**>, *because* <additional rationale **r**>

Step 5: Evaluate ADR. To verify the feasibility of the ADR in each sprint, the EC-ADR checklist (Zimmermann, 2021) will be used in group discussions to improve the traceability of design decisions and emphasize the importance of team consensus in DCD. In this step, the development team goes through the five elements of the checklist to decide whether the particular ADR is aligned with the stated requirements or not. In the latter case, the team will highlight possible concerns against the solution.

Step 6: Evaluate ADR with Design Forces. After analyzing an ADR against the EC-ADR checklist, the development team will verify it with the preliminary list of design forces from step 3 to complete the

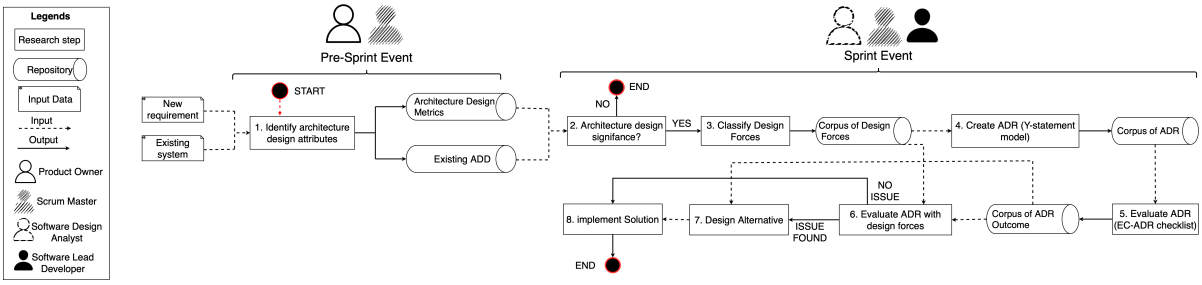

Figure 2: Conceptual framework: main activities for distributed collaborative design decisions.

collaborative software architecture review of the system. This step helps trace the effects of an ADR on the design forces. In case of any constraints, the development team will discuss the potential alternatives previously stated in the ADR and decide accordingly.

Step 7: Design Alternative. This step is executed in case of any issue found in step 6. Based on the available alternatives mentioned in the ADR, the development team will update the suitable solution for the software system.

Step 8: Design Solution. At the end of each sprint, the finalized ADR is shared with the core development team to follow up with the implementation of the corresponding use case.

5 ILLUSTRATING EXAMPLE

To concretely show the potential of our conceptual framework, we used the Fitness Paradise App (FPA) case study (Jolak et al., 2020). FPA is a mobile app where registered users can book facilities/activities in a fitness center, visualize bookings, and share performance with other registered members. The mobile app is designed using the Model-View-Controller (MVC) framework (Krasner and Pope, 1988). We worked in a distributed environment to produce ADDs for the FPA case study using the proposed conceptual framework, as shown in Figure 3.

5.1 First Phase: Pre-Sprint Event

Step 1: Problem Analysis. The product owner and the scrum master elicit three controller components for FPA, i.e., `BookingController`, `ActivityController`, and `PerformanceTracker`. Here, we will focus on `BookingController`.

Action on Framework. To identify the architecture design attributes for `BookingController`, the product owner and the scrum master elicit six use cases (`changeBooking`, `cancelBooking`, `getActivity`, `getFacility`, `managePayment`, `sendInvoice`), the relation of these use cases with other components, and the architecture drivers of the system. In this example, we will focus on `changeBooking`.

Artifact. This phase provides the sprint backlog and the software design specification document, which include a detailed description of all the elicited architecture design metrics extracted from three controller components of the application.

5.2 Second Phase: Sprint Event

Step 2: Check Architecture Relevance. The development team collaboratively analyzes the architecture design metrics of `changeBooking` to identify the design elements for the DCD decisions.

Action on Framework. The following architecture design metrics are identified for `changeBooking`: the registered user uses data from three entities named Booking, Facility, and Activity to manage data, logic, and rules of the applications. The dependency between entities and the `BookingController` class motivates the creation of ADDs for `changeBooking`.

Artifact. A structural design description diagram as an architecture artifact to visualize dependency between the controller classes.

Step 3: Elicit architecture presentation. The software design analyst and the lead developer collaboratively elicit the design forces for `changeBooking` to build the ADR record. The design forces will help evaluate the completeness of the ADR record during the team discussion.

Action on Framework. The design forces for `changeBooking` are classified using six categories (i.e., functional, non-functional, third-party technology, design principles/guidelines, technology selection/implementation, and deployment). `BookingController` is tightly coupled with the `Booking`, `Facility`, and `Activity` classes of the `Model` component; thus, priority is given to three design forces. The output is a list of potential forces for `changeBooking` to create the design record.

Artifact. The list of design forces is elicited for the `changeBooking` to create ADR. For exam-

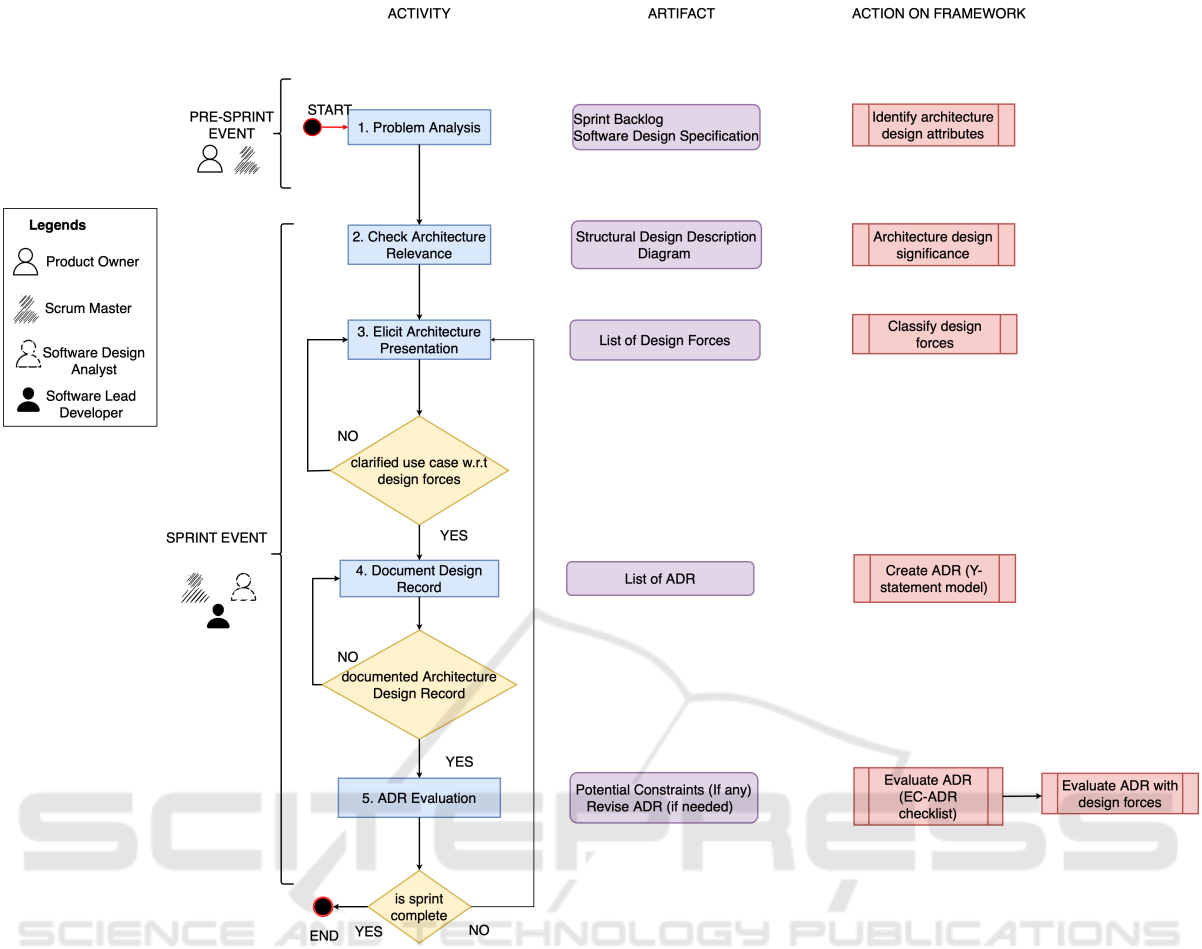

Figure 3: Activities of the proposed framework in the illustrating example.

ple, in the functional design force to implement `changeBooking`, concurrency handling is needed so that multiple users can access data at the same time.

Step 4: Document Design Record. Based on the design forces, the software design analyst uses the Y-statement model to form the ADR.

Action on framework. The resulting ADR is *In the context of* `changeBooking`*, facing the need to keep bookings against members, we decided that* `BookingController` *updates the* `Booking` *components, and neglected* `BookingController` *updates* `Activity` *component or* `BookingController` *updates* `Performance` *component, to achieve functional, non-functional, third-party technology, design principles and guideline, technology selection/implementation and deployment design forces, accepting tight-coupling with* `Model` *components.*

Artifact. The documentation for `changeBooking` ADR is stored and shared with the development team.

Step 5: ADR Evaluation. The development team scrutinizes `changeBooking` ADR to review the design forces. The group discussion aims to: 1) understand the decision context and the assessment circumstances and 2) discuss whether the design forces support ADR or outweigh the forces against it.

Action on Framework. The team uses the following EC-ADR checklist to verify the feasibility of `changeBooking` ADR: 1) Are we confident enough that the design will work? (Evidence), 2) Did we decide between at least two options and compare them systematically? (Criteria), 3) Did we discuss this enough with the team and find a common view? (Agreement), 4) Did we capture the decision outcome and share the design record? (Document), 5) Did we decide when to review and revise the decision if needed? (Revisit/Review).

Artifact. In case of disagreements, potential alternatives will be used as a candidate solution. As we did not find any issue with `changeBooking` ADR record, the design alternative activity of the framework is not performed.

5.3 Discussion

The illustrating example presented in this section shows the effectiveness of the proposed conceptual framework and clarifies how the framework works and its potential benefits for using DCD activities to emphasize the design discussion within the distributed team. Future studies are necessary for empirical evaluation and validation of the framework in real-time environments to identify the advantages of the proposed approach compared to other existing approaches. The framework can be applied in different types of projects and contexts (e.g., academia and software organizations). Therefore, we plan to conduct empirical studies in different contexts, specifically in software organizations willing to improve their DCD practices and in academic courses to prepare students for DCD.

6 CONCLUSION AND FUTURE WORK

In this paper, we introduced a conceptual framework to support Distributed Collaborative Design. The framework leverages state-of-the-art models and new concepts of design decision-making process that are suitable for a distributed environment. The framework helps the distributed teams to perform the activities to build and evaluate ADD records and to decrease the collaboration barriers for participating in the design thinking and decision-making process.

In future, we will implement the framework in a tool that guides the team for completing the framework steps. To do so, we are planning to re-design and integrate the proposed framework in an existing ADD tool to improve the DCD. Then, we will conduct empirical analysis to evaluate the efficiency of the tool in different context (i.e., academia and software software organisations). The results from the empirical analysis will be used to make adjustments accordingly and to proceed with the release of the proposed tool.

REFERENCES

Adil, M., Fronza, I., and Pahl, C. (2022). Software design and modeling practices in an online software engineering course: The learners' perspective. In *CSEDU (2)*, pages 667–674.

Alexeeva, Z., Perez-Palacin, D., and Mirandola, R. (2016). Design decision documentation: A literature overview. In *European Conference on Software Architecture*, pages 84–101. Springer.

Babar, M. A. and Capilla, R. (2008). Capturing and using quality attributes knowledge in software architecture evaluation process. In *2008 First Intl. Workshop on Managing Requirements Knowledge*, pages 53–62.

Babar, M. A., Dingsøyr, T., Lago, P., and Van Vliet, H. (2009). *Software architecture knowledge management*. Springer.

Borrego, G., Morán, A. L., Palacio Cinco, R. R., Rodríguez-Elias, O. M., and García-Canseco, E. (2017). Review of approaches to manage architectural knowledge in agile global software development. *IET Software*, 11(3):77–88.

Bosch, J. (2004). Software architecture: The next step. In *Europ. Conf. on Soft. Architecture*, pages 194–199. Springer.

Capilla, R., Nava, F., Pérez, S., and Dueñas, J. C. (2006). A web-based tool for managing architectural design decisions. *ACM SIGSOFT soft, eng. notes*, 31(5).

Chikofsky, E. and Cross, J. (1990). Reverse engineering and design recovery: a taxonomy. *IEEE Software*, 7(1):13–17.

Cico, O., Jaccheri, L., Nguyen-Duc, A., and Zhang, H. (2021). Exploring the intersection between software industry and software engineering education - a systematic mapping of software engineering trends. *J. of Systems and Software*, 172:110736.

Digital AI (2021). 15th annual state of agile report | digital.ai. https://digital.ai/resource-center/analyst-reports/state-of-agile-report. Accessed: Feb. 2023.

Eris, O., Martelaro, N., and Badke-Schaub, P. (2014). A comparative analysis of multimodal communication during design sketching in co-located and distributed environments. *Design Studies*, 35(6):559–592.

Fronza, I., Corral, L., Wang, X., and Pahl, C. (2022). Keeping fun alive: An experience report on running online coding camps. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*, ICSE-SEET '22, page 165–175, New York, NY, USA. Association for Computing Machinery.

Hesse, T.-M., Kuehlwein, A., and Roehm, T. (2016). Decdoc: A tool for documenting design decisions collaboratively and incrementally. In *Intl. Workshop on Decision Making in Software ARCH*, pages 30–37. IEEE.

Jansen, A. and Bosch, J. (2005). Software architecture as a set of architectural design decisions. In *5th Working IEEE/IFIP Conf. on Software Architecture*.

Jolak, R., Savary-Leblanc, M., Dalibor, M., Wortmann, A., Hebig, R., Vincur, J., Polasek, I., Le Pallec, X., Gérard, S., and Chaudron, M. R. (2020). Software engineering whispers: The effect of textual vs. graphical software design descriptions on software design communication. *Empirical Software Engineering*, 25.

Kemmis, S. and McTaggart, R. (2007). Communicative action and the public sphere. *The Sage handbook of qualitative research*, 3:559–603.

Krasner, G. E. and Pope, S. T. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Object oriented programming*.

Kruchten, P., Capilla, R., and Duenas, J. C. (2009). The decision view's role in software architecture practice. *IEEE software*, 26(2):36–42.

Kruchten, P., Obbink, H., and Stafford, J. (2006). The past, present, and future for software architecture. *IEEE software*, 23(2):22–30.

Lous, P., Kuhrmann, M., and Tell, P. (2017). Is scrum fit for global software engineering? In *2017 IEEE 12th Intl. Conf. on Global Software Engineering (ICGSE)*, pages 1–10. IEEE.

Lytra, I., Carrillo, C., Capilla, R., and Zdun, U. (2020). Quality attributes use in architecture design decision methods: research and practice. *Computing*, 102(2):551–572.

Lytra, I., Tran, H., and Zdun, U. (2013). Supporting consistency between architectural design decisions and component models through reusable architectural knowledge transformations. In *Europ. Conf. on Soft. Architecture*, pages 224–239. Springer.

Muccini, H. and Rekha, S. (2018). Group decision-making in software architecture: A study on industrial practices. *Information and software tech.*, 101:51–63.

Nowak, M. and Pautasso, C. (2013). Team situational awareness and architectural decision making with the software architecture warehouse. In *Europ. Conf. on Software Architecture*, pages 146–161. Springer.

Parizi, R., Prestes, M., Marczak, S., and Conte, T. (2022). How has design thinking being used and integrated into software development activities? a systematic mapping. *J. of Systems and Software*.

Portillo-Rodríguez, J., Vizcaíno, A., Piattini, M., and Beecham, S. (2012). Tools used in global software engineering: A systematic mapping review. *Information and Software Technology*, 54(7):663–685.

Qin, Z., Zheng, X., and Xing, J. (2008). *Introduction to software architecture*. Springer.

Ralph, P., Baltes, S., Adisaputri, G., Torkar, R., Kovalenko, V., Kalinowski, M., Novielli, N., Yoo, S., Devroey, X., Tan, X., Zhou, M., Turhan, B., Hoda, R., Hata, H., Robles, G., Fard, A. M., and Alkadhi, R. (2020). Pandemic programming. *Empirical Software Engineering*, 25(6).

Razavian, M., Paech, B., and Tang, A. (2019). Empirical research for software architecture decision making: An analysis. *J. of Systems and Software*, 149:360–381.

Rice, D. J., Davidson, B. D., Dannenhoffer, J. F., and Gay, G. K. (2007). Improving the effectiveness of virtual teams by adapting team processes. *Computer Supported Cooperative Work (CSCW)*, 16(6):567–594.

Rueckert, J., Burger, A., Koziolek, H., Sivanthi, T., Moga, A., and Franke, C. (2019). Architectural decision forces at work: experiences in an industrial consultancy setting. In *Proc. of the 27th ACM Joint Meeting on Europ. Soft. Eng. Conf. and Symp. on the Foundations of Soft. Eng.*, pages 996–1005.

Safin, S., Juchmes, R., and Leclercq, P. (2012). Use of graphical modality in a collaborative design distant setting. *Work*, 41(1):3484–3493.

Sawyer, R. K. and DeZutter, S. (2009). Distributed creativity: How collective creations emerge from collaboration. *Psychology of aesthetics, creativity, and the arts*, 3(2):81.

Sievi-Korte, O., Richardson, I., and Beecham, S. (2019). Software architecture design in global software development: An empirical study. *J. of Systems and Software*, 158.

Smite, D., Moe, N. B., Klotins, E., and Gonzalez-Huerta, J. (2021). From forced working-from-home to working-from-anywhere: Two revolutions in telework. *arXivpreprint*.

Tofan, D., Galster, M., and Avgeriou, P. (2013). Difficulty of architectural decisions–a survey with professional architects. In *Europ. Conf. on Soft. Architecture*, pages 192–199. Springer.

Tofan, D., Galster, M., Avgeriou, P., and Schuitema, W. (2014). Past and future of software architectural decisions–a systematic mapping study. *Information and Software Technology*, 56(8):850–872.

Vallon, R., da Silva Estácio, B. J., Prikladnicki, R., and Grechenig, T. (2018). Systematic literature review on agile practices in global software development. *Information and Software Technology*, 96:161–180.

Van Heesch, U., Avgeriou, P., and Hilliard, R. (2012). Forces on architecture decisions - a viewpoint. In *Joint Working IEEE/IFIP Conf. on Software Architecture and Europ. Conf. on Software Architecture*, pages 101–110.

Venters, C. C., Capilla, R., Betz, S., and etal (2018). Software sustainability: Research and practice from a software arch. viewpoint. *J. of Systems and Software*, 138.

Weinreich, R., Groher, I., and Miesbauer, C. (2015). An expert survey on kinds, influence factors and documentation of design decisions in practice. *Future Generation Computer Systems*, 47:145–160.

Yang, Z., Xiang, W., You, W., and Sun, L. (2021). The influence of distributed collaboration in design processes: an analysis of design activity on information, problem, and solution. *Intl. J. of Technology and Design Education*, 31(3):587–609.

Zdun, U., Capilla, R., Tran, H., and Zimmermann, O. (2013). Sustainable architectural design decisions. *IEEE Software*, 30(6):46–53.

Zimmermann, O. (2021). ADR = Any Decision Record? Architecture, Design and Beyond. https://ozimmer.ch/practices/2021/04/23/AnyDecisionRecords.html. Accessed: Nov. 2022.

Zimmermann, O., Wegmann, L., Koziolek, H., and Goldschmidt, T. (2015). Architectural decision guidance across projects - problem space modeling, decision backlog management and cloud computing knowledge. In *IEEE/IFIP Conf.on Software Architecture*, pages 85–94.