UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA

Thiago Henrique Ferreira Gomes

The software architecture challenges in Agile Distributed Software Development environments

Recife

2023

Thiago Henrique Ferreira Gomes

The software architecture challenges in Agile Distributed Software Development environments

Trabalho apresentado ao Programa de Pós-graduação em Informática Aplicada da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação

Orientador (a): Prof. Dr. Marcelo Luiz Monteiro Marinho

Recife

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

G633t GOMES, THIAGO HENRIQUE FERREIRA
The software architecture challenges in Agile Distributed Software Development environments / THIAGO HENRIQUE FERREIRA GOMES. - 2023.
111 f. : il.

Orientador: Marcelo Luiz Monteiro Marinho.
Inclui referências e apêndice(s).

Dissertação (Mestrado) - Universidade Federal Rural de Pernambuco, Programa de Pós-Graduação em Informática Aplicada, Recife, 2023.

1. Software Architecture. 2. Distributed Software Development. 3. Communication. 4. Agile Practices. I. Marinho, Marcelo Luiz Monteiro, orient. II. Título

CDD 004

Thiago Henrique Ferreira Gomes

The software architecture challenges in Agile Distributed Software Development environments

Dissertation submitted to the Applied Informatics postgraduate program (PPGIA) from Federal Rural University of Pernambuco (UFRPE), as part of requirements to obtain the Master of Science degree in Applied Informatics. The dissertation was approved by unanimous decision in a public session on August 30th, 2023.

EXAMINATION BOARD

____

Advisor: Prof. Dr. MARCELO LUIZ MONTEIRO MARINHO
Federal Rural University of Pernabuco- UFRPE

____

Prof Dr. RICARDO ANDRE CAVALCANTE DE SOUZA
Federal Rural University of Pernabuco - UFRPE

____

Prof. Dr. IVALDIR HONÓRIO DE FARIAS JÚNIOR
University of Pernambuco - UPE

Dedico aos meus queridos familiares, pelo apoio incondicional, paciência e incentivo ao longo desses anos de estudo. Vocês foram a minha base e fonte de amor e inspiração. Ao meu orientador, Marcelo Marinho, pelo conhecimento transmitido, pelas valiosas orientações e pela confiança depositada em meu trabalho.

ACKNOWLEDGEMENTS

Dedico esta dissertação de mestrado a todas as pessoas que estiveram ao meu lado durante essa intensa jornada acadêmica.

À minha família, que sempre me apoiou incondicionalmente, compartilhando das minhas alegrias e me fortalecendo nos momentos de desafio, principalmente meus pais, Gilvson e Edileide e a minha irmã Thayná e minha avó Suely. Aos familiares que não estão mais presentes entre nós mas que de uma forma contribuíram para minha formação pessoal e profissional, meus avós José e Edite e Gildemar e meu tio Roberto. Obrigado pelo amor, compreensão e pelo constante incentivo ao meu crescimento pessoal e profissional.

Aos meus amigos, que foram verdadeiros pilares de apoio ao longo desses anos. Obrigado pela paciência, pelas palavras de encorajamento e pela companhia nos momentos de descontração que aliviaram o peso das responsabilidades acadêmicas.

Ao meu orientador, Marcelo Marinho, pelo valioso suporte, orientação e conhecimento compartilhado ao longo desta pesquisa. Sua expertise e dedicação foram fundamentais para o desenvolvimento deste trabalho.

Aos professores e profissionais da minha área de estudo, cujas contribuições enriqueceram meu conhecimento e ampliaram minha visão sobre o tema. Obrigado por compartilharem suas experiências e me inspirarem a buscar sempre a excelência na minha área de atuação.

Aos participantes da pesquisa, cuja disponibilidade e interesse em contribuir foram essenciais para a realização deste estudo. Agradeço por compartilharem suas perspectivas e experiências, tornando possível a obtenção de dados relevantes e significativos.

A todos que, de alguma forma, torceram por mim e me motivaram a seguir em frente, meu mais sincero agradecimento. Esta conquista não seria possível sem o apoio e encorajamento de cada um de vocês.

Que esta dissertação seja uma pequena retribuição por todo o carinho e suporte que recebi ao longo desta jornada. Compartilho essa conquista com vocês e espero que ela possa inspirar e contribuir para o avanço do conhecimento em nossa área.

Obrigado a todos por fazerem parte desta caminhada e por tornarem esta realização ainda mais especial.

RESUMO

À medida que o cenário de desenvolvimento de software se globaliza, a comunicação efetiva desempenha um papel fundamental na garantia do sucesso das equipes de desenvolvimento distribuídas. Esta dissertação explora a influência da arquitetura de software na comunicação em ambientes de desenvolvimento distribuído de software. Através de uma abordagem abrangente que inclui um mapeamento sistemático da literatura e um estudo de caso, buscamos compreender e analisar como as diferentes arquiteturas de software influenciam os padrões e práticas de comunicação dentro de equipes dispersas. Realizamos um mapeamento sistemático para identificar pesquisas existentes sobre a relação entre arquitetura de software e comunicação em desenvolvimento de software distribuído (DSD). Esse mapemanto fornece uma visão abrangente do estado atual do conhecimento na área e destaca práticas-chave, tendências e lacunas de pesquisa. A partir dos *insights* obtidos no mapeamento sistemático, aprofundamos o assunto por meio de um estudo de caso detalhado. Nosso estudo de caso envolve um departamento de uma multinaticional européia que adota desenvolvimento de software distribuído trabalhando em projetos diversos, cada um com características distintas de arquitetura de software. Ao analisar os artefatos de comunicação da empresa, realizar entrevistas e observações, buscamos descobrir de que forma específica a arquitetura de software impacta a eficácia e a eficiência da comunicação. As descobertas tanto do mapeamento sistemático quanto do estudo de caso contribuem para uma compreensão abrangente da complexa relação entre arquitetura de software e comunicação em ambientes de desenvolvimento de software distribuídos. Identificamos os pontos fortes e limitações de diferentes arquiteturas de software na facilitação ou obstáculo à comunicação e colaboração efetivas. Além disso, fornecemos recomendações práticas para projetar arquiteturas de software que melhorem a comunicação em equipes distribuídas, aumentando assim o sucesso geral do projeto. Este trabalho destina-se a beneficiar arquitetos de software, gerentes de projetos e profissionais envolvidos em iniciativas de DDS. Ao obter *insights* sobre o impacto da arquitetura de software na comunicação, as organizações podem tomar decisões informadas para otimizar seus processos de desenvolvimento, melhorar a colaboração da equipe e aprimorar os resultados dos projetos em ambientes de desenvolvimento de software distribuído.

Palavras-chaves: Arquitetura de Software. Desenvolvimento de software distribuido. Comunicação. Práticas Ágeis.

ABSTRACT

As the software development landscape becomes increasingly globalized, effective communication plays a pivotal role in ensuring the success of distributed development teams. This work explores the impact of software architecture on communication in Distributed Software Development (DSD) environments. We aim to understand and analyze how different software architectures influence communication patterns and practices within globally dispersed teams through a comprehensive approach that includes a systematic literature mapping and a case study. We conduct a systematic mapping to identify existing research on the relationship between software architecture and communication in global software development. This mapping provides a comprehensive overview of the current state of knowledge in the field and highlights key practices, trends, and research gaps. Building upon the insights gained from the systematic mapping, we then delve deeper into the subject through a detailed case study. Our case study involves a department from a multinational European company that adopts distributed software development teams working on diverse projects, each with distinct software architecture characteristics. We aim to uncover how software architecture impacts communication effectiveness and efficiency by analyzing communication artifacts, interviews, and observations. The systematic mapping and case study findings contribute to a comprehensive understanding of the complex relationship between software architecture and communication in DSD environments. We identify the strengths and limitations of different software architectures in facilitating or hindering effective communication and collaboration. Moreover, we provide practical recommendations for designing software architectures that enhance communication in distributed teams, improving overall project success. This work is intended to benefit software architects, project managers, and practitioners involved in DSD initiatives. By gaining insights into the impact of software architecture on communication, organizations can make informed decisions to optimize their development processes, improve team collaboration, and enhance project outcomes in distributed software development environments.

Keywords: Software Architecture. Distributed Software Development. Communication. Agile Practices.

LIST OF FIGURES

Figure 1 – Method overview 30
Figure 2 – Selection process 33
Figure 3 – Systematic Mapping - Publications by year 42
Figure 4 – Communication and Decoupled-components mindmap 43
Figure 5 – Agile principles and Communication 45
Figure 6 – Architectural-centric development and communication 48
Figure 7 – Losely coupled components and DSD 49
Figure 8 – Survey - Answers by Country 50
Figure 9 – Survey - Answers by Role 51
Figure 10 – Survey - Cultural Diversity 51
Figure 11 – Survey - Years of experience 52
Figure 12 – Survey - Practices Adoption 53
Figure 13 – Survey - Architectural rules on distributed teams 54
Figure 14 – Survey - Aspect impact over communication 55
Figure 15 – Survey - Software Architecture over distributed teams communication . . . 57

LIST OF TABLES

Table 1 – The Selection Criteria 32
Table 2 – Papers by engine. 33
Table 3 – Interviewees profile 38
Table 4 – Survey profile 40
Table 5 – Standard Deviation - SQ1 53
Table 6 – Standard Deviation - SQ2 54
Table 7 – Standard Deviation - SQ3 55
Table 8 – Standard Deviation - SQ4 to SQ17 56

ACRONYMS

API	Application Programming Interface
DDD	Domain-Driven Design
EDA	Event-driven Architecture
gRPC	Google Remote Procedure Call
REST	Representational State Transfer
RPC	Remote Procedure Call
SOA	Service-Oriented Architecture

CONTENTS

1 INTRODUCTION . . . 15

1.1 MOTIVATION . . . 16

1.2 OBJECTIVES . . . 17

1.3 SPECIFIC OBJECTIVES . . . 17

2 THEORETICAL REFERENCE . . . 19

2.1 SOFTWARE ARCHITECTURE . . . 19

2.1.1 Software Architecture Design . . . 19

2.2 ARCHITECTURAL RULES AND DESIGN . . . 20

2.2.1 Application Programming Interface . . . 20

2.2.2 Domain-Driven Design . . . 20

2.2.3 Event-Driven Architecture . . . 20

2.2.4 Microservices . . . 21

2.2.5 Model-Driven Design . . . 21

2.2.6 Service-Oriented Architecture . . . 21

2.2.7 REST/RESTful . . . 22

2.2.8 Component-Based Architecture . . . 23

2.3 DISTRIBUTED SOFTWARE DEVELOPMENT (DSD) . . . 24

2.4 AGILE SOFTWARE DEVELOPMENT . . . 24

2.4.1 Pair Programmming . . . 25

2.4.2 Automated Testing . . . 25

2.4.3 Continuous Integration . . . 26

2.4.4 Refactoring . . . 26

2.4.5 Test-Driven Development . . . 26

2.4.6 Coding Standards . . . 27

2.4.7 Continuous Delivery . . . 27

2.5 SOFTWARE ARCHITECTURE AND AGILE PRACTICES . . . 28

2.6 CLOSING REMARKS . . . 29

3 RESEARCH METHOD . . . 30

3.1 SYSTEMATIC LITERATURE MAPPING . . . 31

3.1.1 Research questions . . . 31

3.1.2 **Definition of inclusion and exclusion criteria** **31**
3.1.3 **Search string** **31**
3.1.4 **Document selection** **33**
3.1.5 **Data Extraction and Analysis** **33**
3.2 CASE STUDY 34
3.2.1 **Case study design** **35**
3.2.1.1 *Objective* *35*
3.2.1.2 *The case* *35*
3.2.1.3 *Theoretical Basis* *36*
3.2.1.4 *Research Questions* *36*
3.2.1.5 *Procedures* *36*
3.2.1.6 *Selection Strategy* *37*
3.2.2 **Preparation to collect data** **37**
3.2.3 **Gathering evidence** **38**
3.2.4 **Analyzing the collected data** **41**
3.2.5 **Reporting** **41**
3.3 CLOSING REMARKS 41
4 **RESULTS** **42**
4.1 SYSTEMATIC LITERATURE MAPPING (SLM) RESULTS 42
4.1.1 **Microservices and Communication** **43**
4.1.2 **Architectural-centric development and performance on distributed teams** **45**
4.1.3 **Agile principles and DSD communication** **47**
4.1.4 **Losely coupled components and communication challenges on DSD** **49**
4.2 CASE STUDY RESULTS 50
4.2.1 **Survey** **50**
4.2.2 **Interviews** **57**
4.3 CLOSING REMARKS 67
5 **DISCUSSION** **68**
5.1 SYSTEMATIC MAPPING 68
5.1.1 **How software architecture design impacts the DSD environment?** . **68**
5.1.2 **Is there any architectural design that can positively impact the DSD environment?** **70**

5.2 CASE STUDY . . . 71
5.2.1 Architecture . . . 71
5.2.1.1 Coding standards . . . 72
5.2.1.2 Adopting archetypes . . . 72
5.2.1.3 Decoupled Architecture . . . 73
5.2.1.4 Team independency . . . 75
5.2.1.5 Guidelines . . . 75
5.2.2 Communication . . . 76
5.2.2.1 Communication Facilitator . . . 76
5.2.2.2 Face-to-face meetings . . . 77
5.2.2.3 Less team interaction . . . 77
5.2.2.4 Agile ceremonies and management frameworks . . . 77
5.2.2.5 Alignment meetings . . . 78
5.2.3 Culture . . . 79
5.2.3.1 Learning culture . . . 79
5.2.3.2 Transparency . . . 79
5.2.3.3 Commitment . . . 80
5.3 DESTILING OUR SUPPOSITIONS . . . 80
5.3.1 SP1: Decoupled software architectures are communication enablers and can help to mitigate communication challenges in DSD environments. . . . 80
5.3.2 SP2: An Architectural Design which enables agile practices and follows architectural-centric principles can help to coordinate DSD teams and mitigate communication challenges . . . 81
5.4 CAN ARCHITECTURAL DESIGN, WHICH ENABLES AGILE PRACTICES AND FOLLOWS ARCHITECTURAL-CENTRIC PRINCIPLES, HELP COORDINATE DSD TEAMS AND MITIGATE COMMUNICATION CHALLENGES? . . . 81
5.4.1 Lessons learned . . . 82
5.5 CLOSING REMARKS . . . 83
6 CONCLUSION . . . 84
6.1 LIMITATIONS AND THREATS TO VALIDITY . . . 85
6.2 FUTURE WORKS . . . 86

REFERENCES 88
APPENDIX A – SELECTED PAPERS 96
APPENDIX B – INTERVIEW QUESTIONS 97
APPENDIX C – SURVEY QUESTIONS 98
APPENDIX D – EXECUTIVE SUMMARY 110

1 INTRODUCTION

Organizations operate in an increasingly global environment, opening up new markets and opportunities, leading to changes in economic conditions and the emergence of competing products and services (CAMARA et al., 2022).

This globalization trend is also evident in software engineering (SE), where national markets have evolved into global markets. This shift has created new competition between countries and fostered a more favorable environment for distributed development, aiming to develop software faster and more cost-efficiently (MARINHO; NOLL; BEECHAM, 2018).

Many companies worldwide have adopted international software development to capitalize on the advantages of this global landscape. This approach allows for increased speed through "follow-the-sun" development, reduced labor costs, proximity to new markets, and access to skilled resources. Developed countries often outsource their software development to emerging economies, seeking economic benefits (MARINHO; NOLL; BEECHAM, 2018; JUNIOR et al., 2022).

Distributed Software Development (DSD) refers to a software development approach where a software project is developed by a team of developers located in different geographic locations, rather than in a single physical location. DSD also encompasses outsourcing, where companies contract software development to vendors in their own or other countries (MARINHO; CAMARA; SAMPAIO, 2021; JUNIOR et al., 2022)

However, DSD introduces additional complexity to project management. The physical distance between teams reduces informal communication that helps clarify specifications and resolve ambiguities. Cultural and language differences can lead to misinterpretation of requirements. Furthermore, limited workday overlap reduces opportunities for synchronous communication, further hindering effective collaboration (CAMARA et al., 2020).

DSD challenges can be addressed by minimizing inter-site communication. Researches (SIEVI-KORTE; BEECHAM; RICHARDSON, 2019; YILDIZ; TEKINERDOGAN; CETIN, 2012) suggests that careful task allocation is vital to achieving optimal communication while reducing the need for extensive site connections. This approach streamlines workflow, resulting in easier task completion, fewer meetings, facilitated email exchanges and minimized misunderstandings due to cultural differences. Task allocation and their interconnections are directly derived from software dependencies dictated by the software architecture.

Furthermore, Conway (CONWAY, 1968) asserts that the software architecture mirrors the

organization's communication structure. Therefore, creating a modular architecture that aligns with the organizational structure and available competencies can effectively address DSD challenges and overcome barriers posed by distance. However, software architecture design is a highly complex activity.

Architects must consider various factors, including required technologies, interdependencies between components, resource availability, budget constraints, scheduling, customer requirements, and marketing pressures. When dealing with multiple layers or components, achieving modularity in software design takes a lot of work.

Numerous studies have explored DSD in the context of software project management, development processes, and organizational factors (SIEVI-KORTE; BEECHAM; RICHARDSON, 2019; YILDIZ; TEKINERDOGAN; CETIN, 2012). However, more research is needed on the intersection of DSD and software architecture (SIEVI-KORTE; BEECHAM; RICHARDSON, 2019).

1.1 MOTIVATION

The Software architecture design includes components and interfaces (PERRY; WOLF, 1992) that connect multiple structures (ALI; BEECHAM; MISTRIK, 2010). The software architecture is used as a project coordination tool (AVRITZER et al., 2010; HERBSLEB, 2007), not only for co-located projects but also as a mechanism to delegate tasks and coordinate distributed teams (CLERC; LAGO; VLIET, 2007a; HERBSLEB; GRINTER, 1999).

A well-defined software architecture benefits the global software process, guaranteeing all team members a common language to define tasks and activities, allowing a better understanding of the business domain, regardless of cultural differences (VANZIN et al., 2005).

Software architecture can assume multiple forms, for instance, a layered structure or a type of structured pipeline. It can present interactions like message based (TAYLOR et al., 1996), or service-based following the Service-Oriented Architecture (SOA) principles (NEWCOMER; LOMOW, 2005), the Representational State Transfer (REST) approach (**??**), or apply one of the most recent tendencies, denominated microservices (WOLFF, 2016; MALAVOLTA; CAPILLA, 2017). The microservices approach evolved from the increasing use of cloud computing (QIAN et al., 2009; KULKARNI, 2012) e following the *aaS (*as a Service*) approach.

Through a systematic literature review, Ali et al. (ALI; BEECHAM; MISTRIK, 2010) synthesize concepts that can be applied during the architecture design process. Other studies, like (FAUZI; BANNERMAN; STAPLES, 2010), consider software development and configuration; however, they

only focus on the process perspective. This indicates that a gap exists in research related to software architecture design in the context of distributed software development.

Therefore, this study aims to address this gap by investigating the relationship between communication challenges and software architecture design within DSD. By exploring this connection, we hope to shed light on the unique considerations and potential solutions that can enhance software architecture practices in a distributed development setting.

Based on that we define our research question which will drive our research during this work: *Can architectural Design, which enables agile practices and follows architectural-centric principles, help coordinate DSD teams and mitigate communication challenges?*

1.2 OBJECTIVES

Aligned to defined research questions, this work aims to contribute a deeper understanding of the relationship between software architecture and communication in distributed software development, providing valuable insights and recommendations for improving collaboration and coordination in DSD projects.

1.3 SPECIFIC OBJECTIVES

In order to achieve the overall objective of this work, the following specific objectives are defined:

- Identify the influence of software architecture on communication in distributed software development environments.
- Gain insights into how distributed teams handle communication challenges in real-world situations.
- Identify and explore practices that can be integrated with software architecture to enhance the communication process in DSD settings.
- Extract lessons learned as outcome and build an executive report about it.

To fulfill the objectives of this study, we conducted a systematic mapping to gather evidence and gain insights into the practices employed for mitigating communication challenges in

software architecture within distributed teams. Subsequently, a case study explored how a specific company has embraced architectural practices in their distributed teams. The case study employed a combination of survey and interview techniques. The survey collected opinions regarding the identified practices identified during the mapping phase. At the same time, the interviews provided qualitative data on the interviewees' perspectives regarding the selection of software architecture and its impact on team communication.

The remainder of this study is organized as follows: In chapter 2, we introduce the background regarding the research subjects that explains the research problem. Chapter 3 describes the proposed methodologies and the research questions. Chapters 4 and 4.2 present the results and implications of the systematic mapping executed and the case study, respectively. Then, chapter 5 has the purpose of discussing the findings and limitations. Finally, in Chapter 6, we state some concluding remarks and areas of future research directions.

2 THEORETICAL REFERENCE

This chapter will present the main concepts needed to understand the topic discussed in this work. We also will discuss the related references relevant to this work, connecting the central points defined by every piece and the point out the gaps.

2.1 SOFTWARE ARCHITECTURE

The international standard ISO/IEC/IEEE 42010:2011 (MAY, 2011) defines Software Architecture as “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”. Software Architecture Design comprehends building software elements and the relationship between them in a manner that the connection between these elements drives us to a high-level description. The architecture modeling documents the decisions and aspects that compose the software architecture (SIEVI-KORTE; BEECHAM; RICHARDSON, 2019).

2.1.1 Software Architecture Design

The software architecture modeling includes components and interfaces (PERRY; WOLF, 1992), which interconnect multiple structures (ALI; BEECHAM; MISTRIK, 2010). The software architecture enables project coordination (AVRITZER et al., 2010), for colocated teams and as a mechanism to allocate tasks and coordinate distributed teams (CLERC; LAGO; VLIET, 2007a).

A well-defined software architecture leverages the process of global software, ensuring every team member has a common language to define tasks and activities. Having a common language enables a better understanding of the business domain regarding cultural differences (VANZIN et al., 2005).

Software architecture can assume multiple shapes, such as a layered structure or structured pipeline. It can interact as the message-based architecture (TAYLOR et al., 1996), or service-based following the SOA (NEWCOMER; LOMOW, 2005), the RESTful approach (RICHARDSON; AMUNDSEN; RUBY, 2013), or even one of the recent tendencies, denominated microservices (WOLFF, 2016; MALAVOLTA; CAPILLA, 2017). The microservice approach evolved based on the increasing demand for cloud computing (QIAN et al., 2009; KULKARNI, 2012) and following the

*aaS (as a Service) structure.

2.2 ARCHITECTURAL RULES AND DESIGN

This section will describe some of the architectural rules, designs, and styles necessary to understand the remaining work.

2.2.1 Application Programming Interface

Historically Application Programming Interface (API) have existed since the beginning of personal computers to connect two or more devices. Well-designed APIs can provide the scaffold necessary for rapid innovation consequence of the critical link enablement that this technic supplies. (IBM, 2016)

There are multiple applications for APIs like API-first(BEAULIEU; DASCALU; HAND, 2022), REST(LI, 2011; PAUTASSO, 2009), Remote Procedure Call (RPC) (MICROSYSTEMS, 2009), Google Remote Procedure Call (gRPC) (CHAMAS; CORDEIRO; ELER, 2017; WANG; ZHAO; ZHU, 1993), and so on. The use of each of these techniques depends on the problem that is going to solve.

2.2.2 Domain-Driven Design

Domain-Driven Design (DDD), defined by Eric Evans (EVANS, 2004) as a philosophy to help with the challenges of building software for complex domains. DDD is not just about software structure but also communication and common language.

Some design patterns are widely adopted when following DDD. These practices help solve everyday problems the developers face during development. These practices include, but not limiting it to, Repositories, Services, Aggregates, Factories, and Values Objects. (EVANS, 2014)

2.2.3 Event-Driven Architecture

Event-driven Architecture (EDA) is a design paradigm in which a software component executes in response to receiving one or more event notifications. EDA is more loosely coupled than the client/server paradigm because the component that sends the notification doesn't

know the identity of the receiving components when compiling. (MARÉCHAUX, 2006; CLARK; BARN, 2011)

One of the main characteristics of EDA is the decoupled interaction, where the sender and the recipient of the message do not know each other existence. (MARÉCHAUX, 2006)

2.2.4 Microservices

Microservices is an architectural style for developing software systems that emphasize decomposing complex applications into more minor, loosely coupled services. Each microservice focuses on a specific business capability and operates as an independent component that can be developed, deployed, and scaled independently. This approach promotes flexibility, scalability, and maintainability in large-scale software systems. (FOWLER; LEWIS, 2014; NEWMAN, 2021)

2.2.5 Model-Driven Design

Model-driven design is an approach to software engineering and system design that emphasizes using models to represent and describe the various aspects of a system. It involves creating abstract models that capture the system's desired behavior, structure, and functionality and then using these models as a basis for system development. In model-driven design, the system is typically represented by a set of interconnected models, each focusing on a specific aspect of the system.(EVANS, 2004)

2.2.6 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is a software architectural style that facilitates the design and development of loosely coupled, modular, and reusable software systems. In SOA, applications are composed of individual services that are self-contained and independent of each other, providing specific functionality as a service to other components or applications within a network. (ERL, 1900)

A service in SOA is a self-contained unit of functionality that can be accessed and invoked over a network through well-defined interfaces. These services communicate by exchanging messages, typically using a standardized protocol such as HTTP(BELSHE; PEON,

2015), SOAP(GUDGIN et al., 2003), REST(DOGLIO, 2015), or RESTful(BIEHL, 2016).

2.2.7 REST/RESTful

The REST (Representational State Transfer) (DOGLIO, 2015) is an architectural style approach for designing networked applications that leverage the existing technologies and protocols of the World Wide Web. REST emphasizes scalability, simplicity, and interoperability between systems. It is commonly used for building web services, APIs (Application Programming Interfaces), and distributed systems.

Fundamental principles of the REST architectural style include:

- Client-Server: The system is divided into the client (requesting) and server (responding) components, allowing them to evolve independently. Clients send requests to servers, which process the requests and return responses.
- Stateless: Each request from a client to a server contains all the necessary information for the server to understand and process the request. The server does not maintain any client-specific state between requests. This simplifies server implementation and allows for scalability and fault tolerance.
- Uniform Interface: REST defines a uniform and standardized set of constraints for resource interaction. This includes using a consistent and predictable set of methods (such as GET, POST, PUT, DELETE) to manipulate resources and adhere to resource identification, representation, and manipulation principles.
- Resource-Based: Resources are at the core of REST. Each resource is identified by a unique URI (Uniform Resource Identifier) and can be represented in various formats, such as XML, JSON, or HTML. Clients interact with resources through the standard HTTP methods.
- Stateless Communication: Each request from the client to the server should contain all the necessary information to process the request. The server does not maintain client-specific context, allowing for scalability and easy distribution.

- Caching: Responses from servers can be cached by clients to improve performance and reduce the load on the server. Caching can be enabled by adding appropriate cache control headers to the responses.

REST has gained widespread popularity due to its simplicity, scalability, and wide adoption in web development. It allows different systems to communicate over standard protocols like HTTP, making integrating and interoperating between various software components and platforms more accessible. RESTful is used to identify an API that implements the REST constraints.

2.2.8 Component-Based Architecture

Component-based architecture (RIGBY, 2016; SCHREYER, 2012) is a software development approach focusing on building systems by composing and reusing modular components. In this architectural style, software systems are designed as self-contained, independent components, each responsible for a specific functionality or service.

Components in a component-based architecture are encapsulated modules that expose well-defined interfaces. They can be developed independently, allowing easier maintenance, testing, and reusability. Components can be implemented using various technologies and programming languages, and they communicate with each other through well-defined interfaces and protocols.

Critical characteristics of component-based architecture include:

- Reusability: Components are designed to be reusable, promoting the development of software systems by assembling existing components rather than building everything from scratch. Reusing components saves development time, improves productivity, and reduces potential errors.

- Encapsulation: Components encapsulate their internal implementation details, exposing only the necessary interfaces to interact with other components. This provides a clear separation of concerns and promotes modularity.

- Independent Development and Deployment: Components can be developed, tested, and deployed independently. This allows for parallel development efforts and easier integration of new components or updates into the system.

- Interoperability: Components interact with each other through well-defined interfaces and protocols, enabling interoperability between different components and systems.
- Scalability: Component-based architectures can be easily scaled by adding or removing components based on the changing requirements of the system.

2.3 DISTRIBUTED SOFTWARE DEVELOPMENT (DSD)

Distributed Software Development (DSD) refers to the execution of software projects by teams operating beyond the confines of a single company, often situated in diverse geographical locations (MARINHO et al., 2019). This dispersion can be at a national level, denoting DSD, or on an international scale, which corresponds to the concept of DSD (MARINHO; NOLL; BEECHAM, 2018).

Moreover, the term DSD encompasses several scenarios, including organizations relocating some or all of their software development resources to regions with lower costs or a more abundant talent pool (JUNIOR et al., 2022). It also encompasses organizations dispersing software development teams across multiple countries (CAMARA et al., 2020). An organization might develop software globally for various purposes, such as use, sale, or integration into a company's products (in-sourcing). Alternatively, a company could outsource the software development process to a supplier within the same country or a different one, where the supplier develops the software for the client (outsourcing) (CAMARA et al., 2020).

Irrespective of the distribution model chosen, companies pursuing distributed development anticipate shared advantages, including cost reductions, access to highly skilled global talent, continuous development around the clock, and potentially lower labor expenses (MARINHO et al., 2019). However, these advantages are not without their challenges; organizations often grapple with issues like ineffective communication, cultural disparities, team synchronization, time zone discrepancies, and maintaining control over the development process (MARINHO; NOLL; BEECHAM, 2018).

2.4 AGILE SOFTWARE DEVELOPMENT

The "Agile Movement" first came to light with the Agile Manifesto, published by software consultants and practitioners in 2001 (BECK et al., 2001). The focus was to bring more impor-

tance to the human aspects over the processes during the software development cycle (BECK et al., 2001). The agile methods have much in common, with the same scaffold, but differ by adopted practices. Extreme Programming (XP) (BECK et al., 2001), Scrum (SCHWABER; SUTHERLAND, 2011), and Lean Development (HIGHSMITH; HIGHSMITH, 2002) are examples of agile methods and frameworks. Some methods and frameworks also focus on agile at a large scale, like SAFe® (LEFFINGWELL, 2018), Scrum of Scrums (SoS) (QURASHI; QURESHI, 2014), and The Spotify Model (SALAMEH; BASS, 2020).

Agile Practices are techniques employed by agile methods or frameworks during development. This work will discuss agile methodologies like Extreme Programming (XP), Scrum, Scrum of Scrum, The Spotify Model, and others. The following subsections will describe some practices adopted in these methodologies.

2.4.1 Pair Programmming

Pair programming is a practice adopted by the Extreme Programming (XP) (BECK, 2001) method and means that two people will develop every code published in an application. The original way of doing that was sitting side-by-side and programming as equals. This process is not a teacher-student relationship.

Some researchers (WILLIAMS, 2000; NOSEK, 1998) have shown that pair programming produces better code quality, improves productivity, and contributes to developing communication skills.

The first application for Pair programming was in co-located teams. Still, there are references (BAHETI; GEHRINGER; STOTTS, 2002) showing the efficiency when adopting this practice in distributed teams in code quality and productivity.

2.4.2 Automated Testing

Automated testing refers to the practice of using software tools and frameworks to execute tests automatically without the need for manual intervention. It involves the creation and execution of test scripts or test cases that verify the behavior, functionality, and quality of software applications or systems. (DUSTIN; RASHKA; PAUL, 1999)

2.4.3 Continuous Integration

The SWEBOK (BOURQUE; FAIRLEY et al., 2014) defines Continuous integration (CI) as a common practice in many software development approaches. It is typically characterized by frequent build-test-deploy cycles.

Martin Fowler says that Continuous Integration is *"A software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly."* (FOWLER; FOEMMEL, 2006)

2.4.4 Refactoring

The SWEBOK (BOURQUE; FAIRLEY et al., 2014) defines refactoring as a reengineering technique reorganizing a program without changing its behavior. It seeks to improve a program structure and its maintainability. Refactoring techniques can be used during minor changes.

2.4.5 Test-Driven Development

Test-Driven Development (TDD) (BECK, 2003) is a software development approach that emphasizes writing automated tests before implementing the code. It follows a cyclical process where small code units are iteratively developed, each time preceded by creating a corresponding test case. The cycle typically consists of three steps: "Red," "Green," and "Refactor."

- Red: In this initial step, a test case is written for a specific desired behavior or functionality. The test is intentionally designed to fail since no code has been implemented yet to satisfy the test criteria.

- Green: The next step involves writing the minimal code necessary to make the previously created test pass. The focus is on implementing the code logic required to achieve the desired functionality.

- Refactor: Once the test passes, the code is refactored to improve its design, structure, and readability without altering its behavior. This step ensures the codebase remains

maintainable, adheres to good programming practices, and eliminates duplication or inefficiencies.

- This iterative process of writing tests first, implementing code to pass those tests, and refining the code is repeated continuously throughout the development cycle. The goal is to create a comprehensive suite of tests that continuously validate the correctness of the code and provide a safety net for making changes in the future.

2.4.6 Coding Standards

The ISO/IEC/IEEE International Standard - 12207:2017 (ISO/IEC/IEEE..., 2017) defines coding standards as part of the development process, and defining it will help clarify the implementation path. Coding standards, also known as programming standards or coding conventions, are a set of guidelines and rules that developers follow when writing source code. These standards define the preferred coding style, formatting, and structure for a particular programming language or development environment. They aim to improve code readability, maintainability, and collaboration among developers working on the same codebase.

2.4.7 Continuous Delivery

Continuous Delivery (CD) (HUMBLE; FARLEY, 2010) is a software development practice that aims to enable frequent and reliable releases by automating the software delivery process. It focuses on ensuring that software can be deployed to production environments rapidly, efficiently, and sustainably.

In Continuous Delivery, the software is built, tested, and prepared for release in a streamlined and automated fashion. It involves the following fundamental principles and practices:

- Continuous Integration (CI): Developers frequently integrate code changes into a shared repository, triggering an automated build process. This ensures that code changes are regularly tested for integration issues and conflicts.
- Automated Testing: Extensive automated testing, including unit tests, integration tests, and functional tests, is an integral part of Continuous Delivery. These tests are executed as part of the deployment pipeline to validate the software's functionality, performance, and stability.

- Continuous Deployment: Once the software passes all the automated tests, it can be automatically deployed to production or staging environments without manual intervention. This allows for rapid and frequent releases.
- Configuration Management: The configuration and environment setup needed for deployment are version-controlled and automated. This ensures that deployments are consistent and reproducible across different environments.
- Deployment Pipeline: Continuous Delivery relies on a deployment pipeline, a series of automated steps that take code changes from version control through build, testing, and deployment. The pipeline provides visibility into the status of each stage and enables efficient collaboration and feedback among team members.
- Monitoring and Feedback: Continuous Delivery emphasizes monitoring the software in production and gathering user feedback. This feedback is used to continuously improve the software and inform future development iterations.

Humble (HUMBLE; FARLEY, 2010) also shows the anti-patterns that are consequences of not adopting these principles, like deploying software manually, deploying to a production-like environment only after development is complete, and manual configuration management of production environments.

2.5 SOFTWARE ARCHITECTURE AND AGILE PRACTICES

Some recent studies have reviewed the relationship between agile practices and software architecture.

Yang et al. (YANG; LIANG; AVGERIOU, 2016) present general benefits of combining architecture and agile practices; like architectural evolution, the architecture-agility combination can facilitate the continuous development of architecture, and the same combination can guide architects in the decision-making process of implementing changes in each agile iteration. From the developer's perspective, agile development can deliver early information and help them make design decisions. Another benefit is the rearchitecting cost; architecting in large increments reduces rearchitecting in agile development because rework costs are incurred due to architecture-related defects and overproduction waste from increments.

Nord and Tomayko (NORD; TOMAYKO, 2006) present some value added through architecture-centric activities combined with agile practices; the designing process provides early feedback to identify trade-offs, risks, and return on investment of architectural decisions.

Breivold et al. (BREIVOLD et al., 2010) show a set of empirical studies related to TDD where applying it increases the code quality and reduces by 40% the number of defects compared with the traditional software development process. Regarding architecture, TDD is a technique that encourages simple design, although the effect of TDD on software architecture is an area where the findings are embedded in empirical discoveries.

2.6 CLOSING REMARKS

In this chapter, we laid the foundation by introducing and elaborating upon the fundamental concepts that are necessary for comprehending the forthcoming work. These concepts serve as the bedrock upon which the entire study rests, providing readers with the tools to navigate the intricate terrain ahead. By delving into the intricacies of these concepts now, we pave the way for a this dissertation. Armed with this conceptual toolkit, readers will be able to traverse the intricate discussions and analyses that await, unlocking insights and a more appreciation of the research to follow.

3 RESEARCH METHOD

This work employs a methodological approach combining systematic mapping and a comprehensive case study to achieve its research objectives. The systematic mapping phase involves a literature review to identify and categorize existing studies in the field, providing a structured overview of the research landscape. Subsequently, a case study focuses on a specific context of interest. The case study employs a mixed-methods approach, incorporating in-depth interviews and a survey. Interviews are conducted with key stakeholders and experts to gather qualitative insights, while the survey collects quantitative data from a broader sample, enhancing the findings. The integration of these methods enables a holistic understanding of the research problem, combining rich qualitative insights with statistically significant quantitative data. Figure 1 present an overview of the process applied.

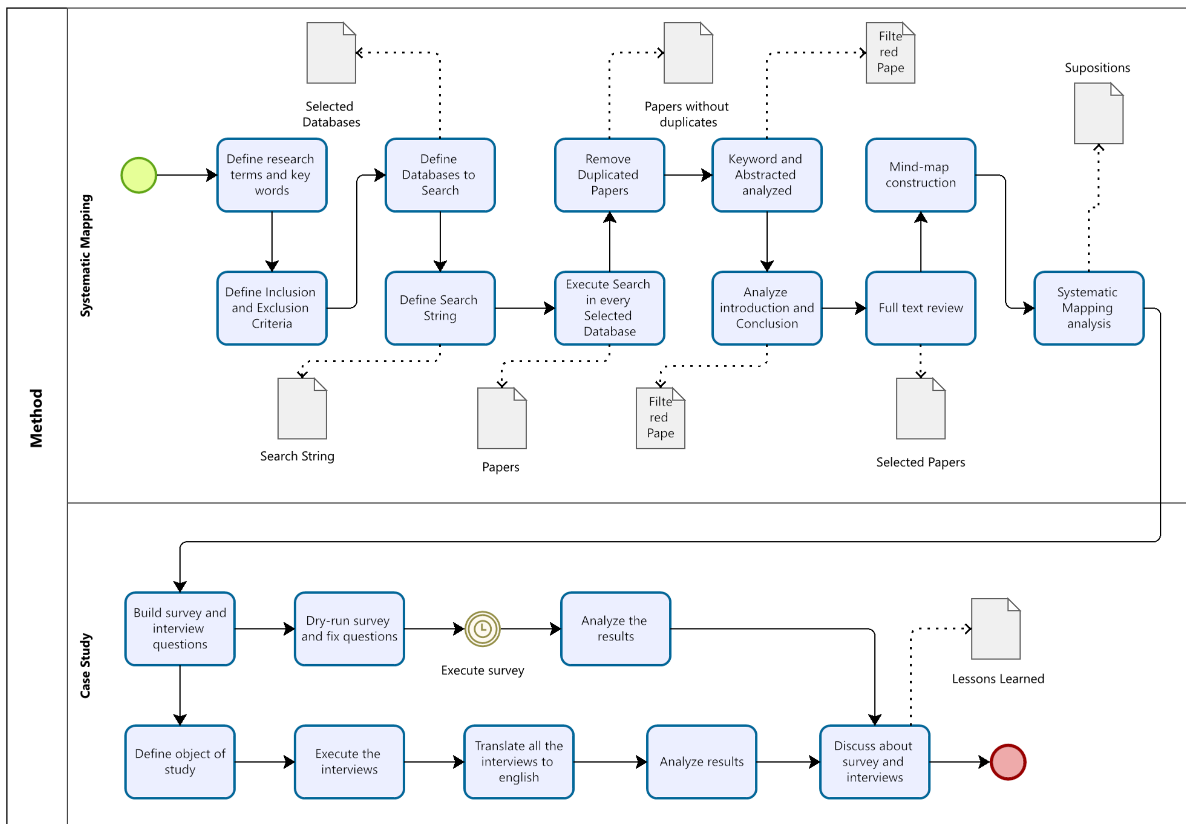

Figure 1 – Method overview

3.1 SYSTEMATIC LITERATURE MAPPING

We adopted a systematic and focused approach to examine the relevant literature in this study. Rather than uncovering every recorded practice, we aimed to select a representative collection of studies to identify recurring themes.

Established SLM guidelines (PETERSEN et al., 2008) recommend that a reviewer carry out the following steps: (i) search for Relevant Studies; (ii) study selection; (iii) data extraction; (iv) results analysis; (v) results synthesis.

We conducted a SLM, following the guidelines proposed by Kitchenham and Charters (KITCHENHAM; CHARTERS, 2007). First, we defined the research questions to guide us during this study. After this, we specify the keywords and their synonyms related to our research topic and use them to build our search string. Next, we selected the target databases and executed the search string. Finally, we started the extraction processing that will be described in more detail in section 3.1.4.

3.1.1 Research questions

We sought to answer the following research questions:

RQ1 How software architecture design impacts the DSD environment?

RQ2 Is there any architectural design that can positively impact the DSD environment?

3.1.2 Definition of inclusion and exclusion criteria

We use comprehensive selection criteria to help us identify more studies. The selection criteria used in this study are described in Table 1.

3.1.3 Search string

We used terms related to software architecture, global software development, development practices, and their synonyms to build our search string. To have a more accurate outcome, we decided to limit our results to publications from 2003. We selected this year considering

Table 1 – The Selection Criteria

Criteria	
Inclusion	▪ Studies that approach using software architecture inside of the DSD environment. ▪ Papers in which the keywords appear on Abstract and/or Author keywords.
Exclusion	▪ Papers are not related to DSD and software architecture design simultaneously. ▪ Studies related to teaching DSD. ▪ Studies which focus is not software architecture design. ▪ Studies not written in English.

the first paper found related to decoupled-resilient software architectures, written by Perrey and Lycett (PERREY; LYCETT, 2003).

To identify a set of relevant papers for our study, we conducted searches using a targeted group of keywords. Our search strategy began with examining top-ranked hits using simple keywords, which the final complex search string may have overlooked. We started with general keywords such as Software Architecture, Microservices, DSD, Agile Methods, Hybrid Methods, and their possible synonyms to cast a wide net.

We used the following boolean search string to ensure that we captured a wide variety of papers:

((“Resilient software architecture” OR “Decoupled software architecture” OR “Resilient software architectures” OR “Decoupled software architectures” OR “Decoupled-resilient software architecture” OR “Decoupled-resilient system architecture” OR “Decoupled-resilient systems architecture” OR “modern software architecture” OR microservices OR “micro services” OR “software architecture”) AND (Devops OR agile OR scrum OR “extreme programming” OR “pair programming” OR hybrid OR “lean development” OR “lean software development” OR SAFe OR “Scaled Agile Framework”) AND (“global software engineering” OR “global software development” OR “distributed software engineering” OR “distributed software development” OR GSE OR GSD OR “distributed team” OR “global team” OR “dispersed team” OR “spread team” OR “virtual team” OR offshore OR outsource OR nearshore))

3.1.4 Document selection

The first step in our selection process involved identifying the relevant databases for our study. We chose to use IEEE, ACM, Scopus, and Springer. We then executed a research string built around the keywords outlined in section 3.1.3, resulting in 3471 papers. After removing duplicates, we were left with 3298 articles. Upon reviewing the titles and abstracts, we narrowed the selection down to 47 papers, which we read in full. We selected 13 reports (Appendix A) as our final set of papers. The entire selection process is illustrated in Figure 2.

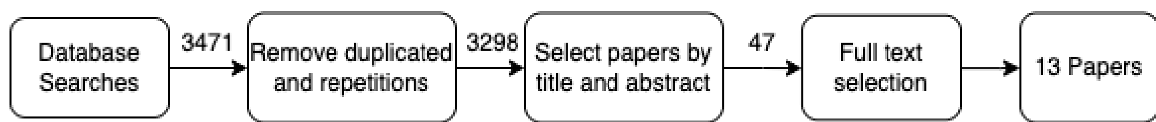

Figure 2 – Selection process

Engine	Selection	Removing Duplicated	Selected	Full-text Selection
ACM	708	682	16	6
Scopus	772	702	11	4
Springer	1371	1314	3	2
IEEE	620	600	17	1
Total	**3471**	**3298**	**47**	**13**

Table 2 – Papers by engine.

3.1.5 Data Extraction and Analysis

Once we had our set of selected papers, we used ATLAS.ti (Scientific Software Development GmbH,) to analyze them. Each study was examined, and we extracted fragments from the papers (quotes) and grouped these fragments into categories (codes) relevant to our research questions. The goal was to understand each author's perspective on software architecture design, communication, and solutions to mitigate challenges in GSD, as presented in Chapter 4.

Following the analysis, we created three mindmaps to visualize the connections between ideas and themes. The mindmaps were organized into three categories: communication, software architecture design, and solutions to GSD challenges. These mindmaps provide a holistic

view of the relationships between various concepts and ideas, which will help us better to understand the authors' perspectives on these topics.

3.2 CASE STUDY

Effective communication is a cornerstone for project success, team cohesion, and innovative problem-solving in modern software development. As software development teams grow in complexity and diversity, the impact of communication on their performance becomes increasingly vital. Thus, undertaking a comprehensive case study within a company offers a unique opportunity to delve deep into the intricate dynamics of communication within software development teams. By closely examining communication patterns, information flow, and collaboration practices, this case study aims to uncover insights that can optimize team productivity, enhance project outcomes, and foster a culture of creativity and knowledge sharing. By investigating real-world scenarios and contextual factors, this research seeks to contribute to the knowledge of software development methodologies and provide actionable recommendations to businesses striving to cultivate efficient communication strategies within their software development teams.

To execute our case study, we followed the process guidelines defined by (RUNESON; HÖST, 2009), which is composed of five major steps:

1. Case study design: define the objects and plan the case study.
2. Preparation to collect data: describe the process of collecting data.
3. Gathering evidence: case study execution with data collection.
4. Analyzing the collected data.
5. Reporting.

In this work, we defined each step mentioned before in a trustworthy manner to build the case study to provide quality and credible work.

The following sections will provide more details about each step described before.

3.2.1 Case study design

Before conducting our case study, we must establish the process to guide our execution. Our plan for the study case is based on (ROBSON, 2002) and consists of six components as described below:

1. **Objective:** Where we tell what we want to achive with the study.
2. **The case:** We describe the unit of analysis and what is studied.
3. **Theoretical Basis:** We describe the references used to build or research questions and which gave origin to this case study.
4. **Research Questions:** The research questions will drive the collection process.
5. **Procedures:** Here, we will explain the strategies adopted to collect data.
6. **Selection Strategy:** Where to seek the data.

The following sections will illustrate each one of these components and explain the adopted process.

3.2.1.1 Objective

The main objective of this case study is to identify practices related to software architecture design that can improve the interaction between team members in a distributed software development environment. To achive this objective, we are going to conduct an exploratory analysis.

3.2.1.2 The case

The unit of analysis of this case study was an international retail company with a presence in all the continents with annual net sales of around 24 million dollars. Its software development staff is distributed in 5 different countries. This company has around 60000 employees worldwide, and about 8.5% are IT professionals. We will analyze a specific department with

around 70 people responsible for developing digital products to improve the product development process. This department has teams in two countries in the same timezone, adopting a near-shore approach.

Regarding the guidelines adopted by this company related to adopting tools, programming languages, and guidelines, we can describe the following guidelines. The main programming languages used include Java, JavaScript, Groovy, Golang, and Python. In terms of tools, we rely on Jira, Bitbucket, Git, Confluence, Jenkins, Kubernetes, and Docker. These guidelines serve as a foundation for our development and collaboration processes.

3.2.1.3 Theoretical Basis

The theoretical basis of this case study is our theoretical reference defined on Chapter 2 and also from the systematic mapping results presented on Chapter 4.

3.2.1.4 Research Questions

In this section, we will present the research questions which drive our investigation process and the procedures adopted in this case study. These research questions result from the first stage of this work, our systematic mapping. We present the two questions in the list below.

- Are decoupled software architectures communication enablers and can help to mitigate communication challenges in DSD environments?
- Architectural Design, which enables agile practices and follows architectural-centric principles, can help to coordinate DSD teams and mitigate communication challenges?

3.2.1.5 Procedures

There are two parts to the execution process in this study. The first part is a semi-structured interview based on the guidelines described by (RUNESON; HÖST, 2009), following the questions described in Appendix B. In this step, we aim to identify, in a broad way, practices and approaches which may improve the dynamics in distributed software development teams. These questions were used to guide our interview, but we were not limited just to them. The second

part is a survey based on the inquiries related in Appendix C following the guidelines defined by (EASTERBROOK et al., 2008), aiming to validate the findings from our interviews.

3.2.1.6 Selection Strategy

The selection strategy adopted in the interviews in this case study was to get people with leadership roles and convenience because we had easy access.

3.2.2 Preparation to collect data

This section will present every aspect behind the questions and alternatives used in both interview and survey during the execution.

As defined in Appendix B, every question adopted in our interview has a code that identifies it; the code structure is a prefix **IQ**, which determines the question used in the interview step, followed by a two-digit sequential number, to differentiate the questions, for example, **IQ01**, representing the first question from our interview protocol. During this work, we will reference these questions using their respective code.

As defined in Appendix C, every question and alternative in our survey has a code that identifies it. This section will use these codes as references to explain the objective after each question and option.

There are three sections in the Survey. The first section is about the processes and practices adopted by the teams and to understand how frequently the teams adopt the techniques daily. In the second part, we aim to collect the respondents' opinions about the impact of software architecture on communication aspects in distributed software development teams. In the last section, we aim to gather demographic information about the subjects to help us to analyze the respondents' characteristics. In this case, the objective is to identify these practices' knowledge and usage and collect opinions about their impact on team communication.

Each survey question and alternative will have a code to help the reference. The questions are coded using the prefix **SQ** and two digits numbers, for instance, **SQ07** representing the seventh question in our survey. The alternatives are coded using the question as a code prefix followed by a two-digit sequential number to differentiate them. For example, the first alternative from question seven will have the code **SQ07A01**. Every code defined can be founded in Appendix C. The following sections will use these code structures when discussing

questions or options.

3.2.3 Gathering evidence

In this section, we will present the evidence collected during the execution. The time aspect and demographic will be present here, for instance, how long we execute our interviews and how long the survey was open to answers.

We executed four interviews, and all were recorded, producing 1 hour and 16 minutes of recorded material. Before starting the interview, we ask for permission from the respondents to record the interview. Table 3 shows the profile of the interviewees. The interviews were conducted in three different languages: Portuguese, English, and Spanish as the interviewee preference. The interviewees were selected by convenience and availability, all of them are from the same department as the researcher who conducted this research.

Code	Role	Years of experience	Location	Interview Language
EA	Solution Architect	$>$ 10 years	Germany	English
EB	Solution Architect	$>$ 10 years	Germany	English
EC	Technical Lead	$>$ 10 years	Spain	Spanish
ED	Technical Lead	$>$ 10 years	Spain	Portuguese

Table 3 – Interviewees profile

To keep the respondents anonymous, we assign each one a code, and from now on, we will refer to them using their respective identification codes.

Interviewee EA is a Solution Architect with a more hands-on approach, his primary responsibility inside the product area is to help multiple teams to design solutions, and he also works as a Product Owner in some groups. He has a high-level role, sometimes coordinating other solution architects.

Interviewee EB is also a Solution Architect with a hands-on role. However, he has a different level of responsibility than interviewee EA; he actuates mainly in a single team and collects requirements from product owners and business clients.

Interviewee EC and ED have similar role. However, in different teams, they are primarily responsible for driving the teams to archive their teams objective, helping team members to remove impediments during their everyday activities, and they also are part of defining objectives for each quarter.

The survey stayed open to answers for six months and was distributed using company email groups limited just to the unit of analysis. All the answers were anonymous, but the respondent could inform any e-mail if they wanted to receive the results from this work.

Table 4 presents the respondents' profile in a high-level view, there you can have a notion about the amount of experience that each one has, the country where they are located, and the role performed in the current team. Each respondent has their own identification code, for the remainder of this work, we will reference the respondents using these codes.

Code	Role	Years of experience	Location
S1	Software Developer	$>$ 10 years	Spain
S2	Software Developer	6-10 years	Spain
S3	Software Developer	$>$ 10 years	Spain
S4	Software Developer	$>$ 10 years	Spain
S5	Software Developer	3-5 years	Spain
S6	Software Developer	$>$ 10 years	Spain
S7	Software Developer	3-5 years	Spain
S8	Software Developer	6-10 years	Spain
S9	Software Developer	3-5 years	Spain
S10	Software Developer	3-5 years	Spain
S11	Team Lead	$>$ 10 years	Spain
S12	Software Architect	6-10 years	Spain
S13	Team Lead	$>$ 10 years	Spain
S14	Software Developer	6-10 years	Spain
S15	Software Architect	$>$ 10 years	Germany
S16	Team Lead	$>$ 10 years	Spain
S17	Software Architect	$>$ 10 years	Germany
S18	Software Architect	$>$ 10 years	Spain
S19	Enterprise Architect	$>$ 10 years	Germany
S20	Software Architect	$>$ 10 years	Germany
S21	IT manager	$>$ 10 years	Germany
S22	Team Lead	$>$ 10 years	Spain
S23	Team Lead	$>$ 10 years	Spain
S24	Team Lead	6-10 years	Spain
S25	Quality Analyst	6-10 years	Spain
S26	Quality Analyst	3-5 years	Spain
S27	Software Developer	6-10 years	Spain
S28	Software Developer	$>$ 10 years	Spain
S29	Quality Analyst	3-5 years	Spain
S30	Quality Analyst	$<$ 1 year	Spain
S31	Software Developer	6-10 years	Spain
S32	Software Developer Engineer in Test	6-10 years	Spain

Table 4 – Survey profile

3.2.4 Analyzing the collected data

As an outcome of the interviews, we had the recordings files, some with just audio and others with audio and video. These recordings were transcribed using a cloud service from Amazon called AWS Transcribe[1], developed to convert speech to text. After having the transcription, we reviewed it and imported the text into Atlas.ti[2] to start the qualitative analysis.

The collected data was analyzed following the guidelines defined by the Grounded Theory (GLASER; STRAUSS; STRUTZEL, 1968). The first step was to code each line identifying concepts and key sentences and then move to categories and sub-categories where the data from each participant will be compared for similarities. The second step is to do an Axial analysis using the defined categories and identify their relationship. The final step is called selective coding, which consist of determine the core coding and methodically relate it with other codes.

In the survey, all the answers were collected using a Google Form[3], and each answer was placed into a spreadsheet to help us generate graphics and understand the relation between the data.

3.2.5 Reporting

We will discuss the collected data in this work's Chapter 4.2.

3.3 CLOSING REMARKS

In this chapter, the methodology employed for this study was elaborated in detail. The systematic mapping approach was meticulously explained, encompassing the criteria utilized for paper selection, data extraction, and quality evaluation. Additionally, the rationale underlying the selection of a pertinent case study was expounded upon, highlighting its alignment with the research objectives and its significance in generating meaningful results. This methodological groundwork lays the groundwork for the ensuing presentation of results, offering readers a clear understanding of the rigor and consideration that underpin the study's outcomes.

[1] https://aws.amazon.com/pt/transcribe/
[2] https://atlasti.com/
[3] https://google.com/forms/about/

4 RESULTS

This chapter presents the culmination of our research endeavor, which seamlessly the systematic mapping methodology with an intensive case study approach. By harnessing the synergy between these two methodologies, we have explored and analyzed the multifaceted landscape of our research domain. In the preceding sections, we unravel the insights garnered through the systematic mapping phase, unveiling the intricate web of existing knowledge, trends, and gaps within the field. Building upon this foundational groundwork, we transition into the core of our investigation - the comprehensive case study. Through a judicious combination of in-depth interviews and a targeted survey, we have delved deep into a specific contextual setting, capturing both qualitative nuances and quantitative patterns. This chapter stands as a testament to the relationship between systematic mapping and the case study, each complementing the other in a pursuit of enriched understanding and findings.

4.1 SYSTEMATIC LITERATURE MAPPING (SLM) RESULTS

In this section, we will present our SLM results, which have been grouped into three main categories: communication, software architecture, and solutions to mitigate challenges in DSD, and their interrelationships. We will refer to the selected papers Appendix A using the SMXX format, where XX represents the paper ID with two digits. We have generated a chronological distribution chart to analyze the selected papers' characteristics (Figure 3).

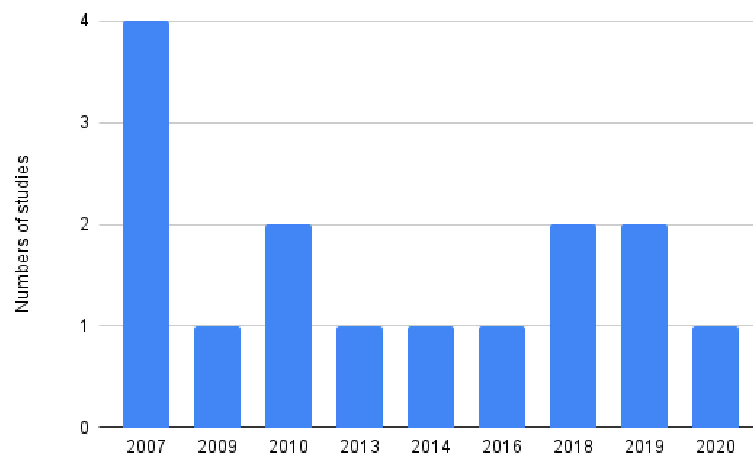

Figure 3 – Systematic Mapping - Publications by year

Transitioning to the forthcoming section showcasing the systematic mapping results, we

embark on a journey of structured exploration. These results, curated through a process, will be presented in a manner that fosters clarity and comprehension. The findings will be organized into four distinct categories, each represented through illustrative mind maps to facilitate an intuitive understanding. These visual aids serve as navigational guides, ushering readers through the intricate web of emerging research trends, gaps, and relationships. As we delve into this section, the synthesis of systematic mapping outcomes into these comprehensive categories aims not only to enhance accessibility but also to provide a holistic view of the knowledge landscape we've unraveled.

4.1.1 Microservices and Communication

On the first mindmap (see Figure 4), the first important aspect is that applying Conway's law combined with decoupled components can help us mitigate communication challenges, which means that the components should reflect the organizational structure (CONWAY, 1968; PETROV; AZALETSKIY, 2023).

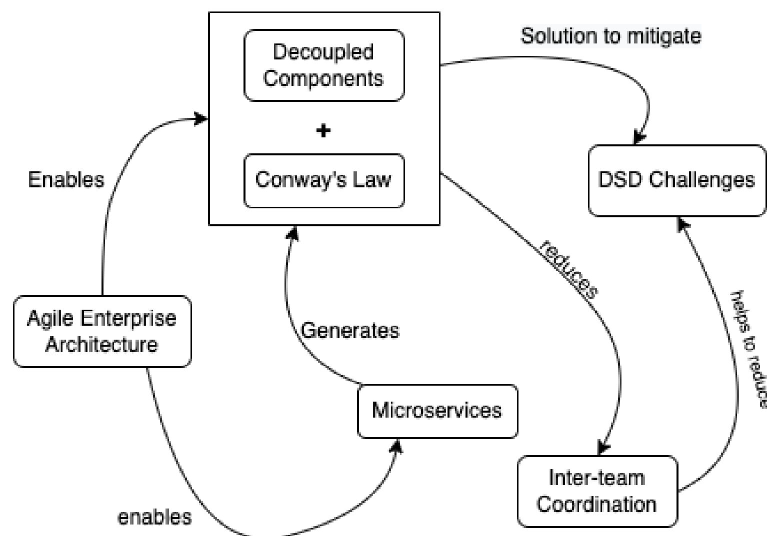

Figure 4 – Communication and Decoupled-components mindmap

Sievi-Korte et al. [SM10] bring that using Conway's law, which states that software architecture will, at some point, reflect the organization structure, with modular architectures could help mitigate many DSD challenges, including communication.

Alzoubi and Gill [SM13] highlight that using AEA as a typical model between the DSD teams could enable communication and decrease misunderstandings and unnecessary contact because software definition and structure are needed. This type of model helps to generate

decoupled components and provides a possibility to coordinate through the component's interfaces. Using this approach allows the teams abroad to build each part separately. Furthermore, Sievi-Korte et al. [SM09] present some references recommending using Conway's law when designing software architecture. These references claim that the more separated the components are, the more likely the organization will be able to develop them successfully on multiple sites.

This improvement is possible by using components interfaces and reducing the need for inter-team communication between distributed teams. Each team should follow the interface definition to build their components and communicate with other teams when any interface modification occurs. Therefore, in these situations, the communication challenges are mitigated by having limited communication.

Regarding inter-site coordination, Alzuobi and Gill [SM13] present that architecture-based development can help us identify highly independent components and use them to divide the development tasks among the distributed teams, decreasing the necessity for inter-site coordination. Mishra and Mishra [SM07] also reinforce that software architecture helps decrease the necessity for communication in a multi-site development project, reducing inter-team communication.

Lenarduzzi and Sievi-Korte [SM06] highlight that microservices architecture ends up in the same environment as global software development teams, which are developing different parts of the same system. They also bring the possibility to overcome communication problems by having a layer on the communication structure that will become a coordinator among the teams. Moreover, microservices carry many complexities, so the development must rely on software architects who can also be the coordinator role. However, having the coordinator has pros and cons, like:

1. *One-level hierarchy*: one person will manage the problems, and they will be the only ones responsible for that. This approach reduces the decision time, although it could result in a non-democratic team. This person also needs to have a good level of expertise.

2. *Two-level hierarchy*: the global coordinator and the leader of each microservice team are two layers of the decision-making chain. It possibility each group to have a representative on each decision, although it could generate problems in synchronizing the communication between the coordinators.

3. *Full democracy*: the decisions are taken after discussion between all team members or

by the most representative from each team. It decreases the possibility of exclusion of the team members, but the discussion will take longer.

Still, on the first mindmap, Agile Enterprise Architecture (AEA) (HERBSLEB; MOITRA, 2015; OVASKA; ROSSI; MARTTIIN, 2003) may enable microservices architecture, which can help the teams achieve the benefits generated by using decoupled components when Conway's law is applied. [SM13]

4.1.2 Architectural-centric development and performance on distributed teams

On the second mindmap (see Figure 5), the main focus is the performance in distributed teams. The main factors impacting the performance focus on architectural aspects, such as centralized architectural modifications, architecture-centric development, agile enterprise architecture, and architectural knowledge management.

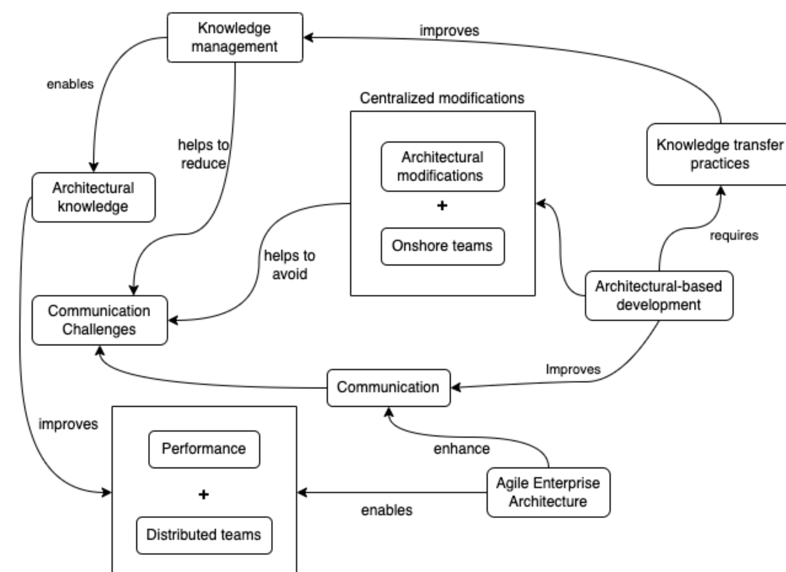

Figure 5 – Agile principles and Communication

Alzoubi and Gill [SM13] present that integrated AEA views could serve as a base or common language that will improve the understanding of the technology point of view and business perspective. They also provide empirical evidence that implementing AEA will enhance the performance of DSD teams by implementing AEA and also bring contrast with using EA for not delivering value.

Regarding knowledge management and communication challenges, Clerc et al. [SM03] present a study of cases where they analyze two organizations and discuss possible solutions

to mitigate some DSD challenges. They found that organizations use a wiki, highly communicative meetings, and subsystem websites to reduce difficulty in exchanging information. Furthermore, both organizations use this strategy when discussing centralized modifications by onshore teams. The first organization has an architecture team that supports various projects and defines general architectural rules for all subsystem teams. If there are some system-specific issues, the subsystem architect should handle them. The second organization has a software engineering process guideline but often deviates from it. They also had an integration team responsible for integrating all the systems at the end, but they needed an architectural compliance verification; due to this, multiple processes co-exist in real-world routine.

The onshore team is responsible for architectural modifications when applying an architectural-centric development approach. The findings show that it helps avoid communication challenges, a concept shown on the mindmap as centralized modifications. Architecture-centric development also improves knowledge management because knowledge transfer practices help reduce communication challenges.

In the relation between knowledge transfer practices and their impact on communication challenges, Urrego et al. [SM01] say that large distances between team members indicate issues related to issuing the resolution, effective communication, the first contact between distributed members, and lack of trust. Kornstädt and Sauer [SM04] highlight that significant communication gaps will sooner or later lead to miscommunications, which brings even more concern, mainly to projects with complex applications. To avoid the source of miscommunication, Kornstädt and Sauer [SM04] also present a set of development processes applied to the organization studied, which could mitigate the communication challenges using feedback loops. Some of these practices are:

1. *Releases* aim to develop new features for the application and make them available as soon as possible. It helps to reduce frequent problems related to outdated specifications.

2. Daily *stand-ups*, a quick meeting to discuss the tasks developed since the last stand-up, a little bit about what everyone plans to do until the next one, and use evenly to spread knowledge about what is going on in the project.

3. *Pair programming* occurs twice a day. Two developers share the same computer, aiming to have common knowledge about nearly every piece of code. During this process, the

developers are exposed to each other criticism every time, and software concepts are constantly a subject of debate.

Regarding the impact of architectural-based development on communication, Kornstädt and Sauer [SM04] show that implementing architectural-based development eases communication by supplying an understanding via one general object of work that all team members use to comprehend. It also helps to establish a basis for verifiable architectural rules and automatically check them, reducing errors and improving implementation reliability. Kornstädt and Sauer [SM05] highlighted that the stakeholders could use this object of work point using general terms and concepts as a common language, facilitating the discussions and arrangements. Furthermore, architecture-based development brings other advantages to the communication aspects, such as task allocation, construction, and record of experience.

Concerning the impact of knowledge transfer practices when applying architectural-based development, Kornstädt and Sauer [SM04] developed a case study where the organization used to execute all the architectural modifications only on an onshore team. The learning curve for offshore developers was remarkably abrupt, whereas supplying good examples like equivalent components implemented by other developers with more experience could significantly enhance the knowledge of the offshore team members.

The findings also show that agile enterprise architecture enables performance in distributed teams and enhances communication, which helps mitigate communication challenges.

Regarding the impact of architectural knowledge on the performance of distributed teams, Clerc [SM02] brings to our attention that architectural knowledge concentrates on architecting as a process to make decisions and is not yet accepted abroad by distributed teams developing software. He also tells us that architectural knowledge needs to address performance as a vital quality criterion. Clerc also points out that the architectural knowledge topic only applies to projects on a multi-site.

4.1.3 Agile principles and DSD communication

On the third mindmap (see Figure 6), the main focus is the impact of agile principles on distributed teams' communication. Alzoubi and Gill [SM11] show that face-to-face communication and daily work projects are practical in small co-located teams. However, the opportunity for these practices is limited in distributed teams. Meanwhile, Alzoubi and Gill also bring that

AEA can be used as a communication enabler beyond that by using it as an integrated shared view.

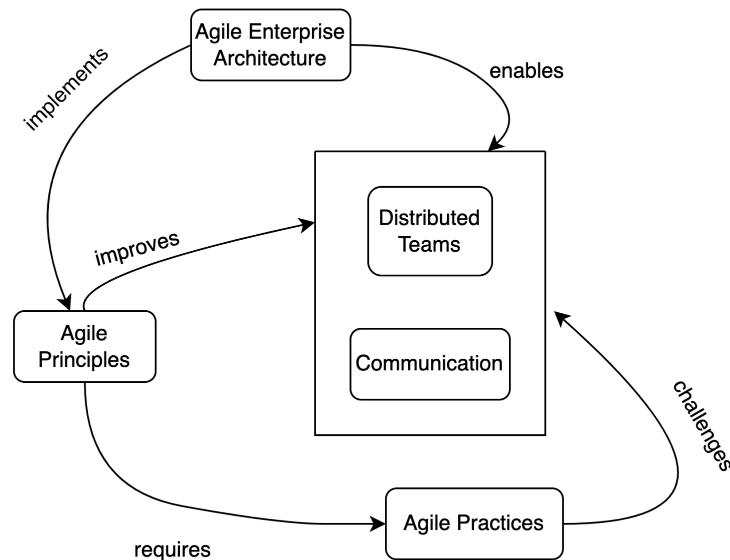

Figure 6 – Architectural-centric development and communication

Regarding applying agile principles with agile practices over the communication aspects in distributed teams, Kornstädt and Sauer [SM04] highlight techniques, like pair programming and daily stand-ups, used by companies to help avoid communication problems by having frequent communication.

When discussing agile practices' challenges over communication teams, Gill and Alzoubi [SM11] tell us that the best result regarding architecture, requirements, and design comes from self-organized teams, and the communication between business people and developers needs to happen daily. However, many barriers challenge the communication between developers' teams and business people, even more when these teams need to develop features inter-dependant features and work simultaneously. Otherwise, using AEA as an integrated shared view may provide a comprehensive picture that can help enhance team communication and overcome the problems related to cultural differences and spoken language. Consequently, it may increase communication effectiveness, indicating that AEA can be used as a communication enabler mechanism. Nevertheless, Alzoubi et al. [SM12] conclude that it is not clear how AEA affects geographically distributed teams.

4.1.4 Losely coupled components and communication challenges on DSD

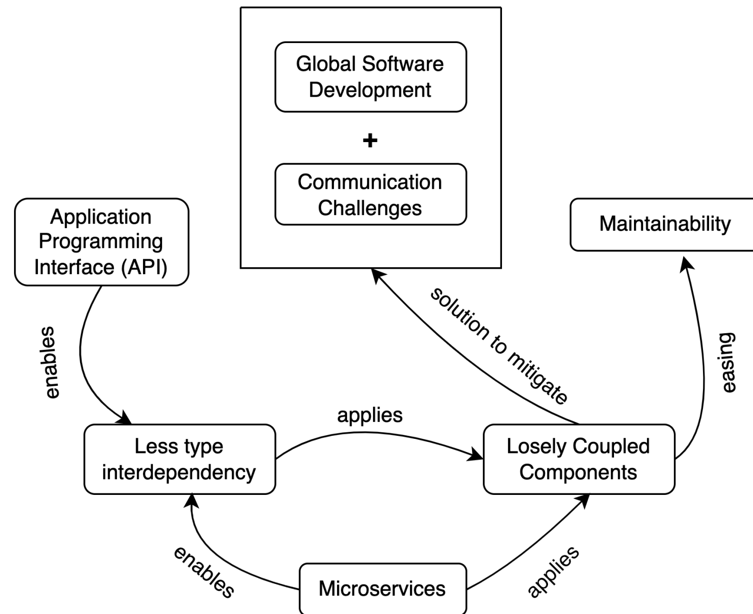

Figure 7 – Losely coupled components and DSD

On the last mindmap (Figure 7), Sauer [SM09] recommends following some SOLID principles (MARTIN, 2017), like the open-closed principle. He also proposes the adoption of the other tenets, like avoidance of type interdependencies, loose coupling, design by contract, and strong cohesion, which are the scaffold behind the understandable software to achieve understandable software on distributed projects regarding the finite opportunities to communicate and the source code becomes the primary basis of knowledge.

Tekinerdogan et al. [SM08] bring to our attention that the most acceptable practices of software architecture strategy constitute loosely coupled components with well-defined contracts. Microservices (NEWMAN, 2021) architectures allow this design strategy and therefore enforce minor type interdependency.

As a possible solution to mitigate communication challenges in a DSD environment, Sievi-Korte et al. [SM10] indicate the use of APIs as a crucial architecting practice. In this context, projects can use APIs to handle interfaces and modules' boundaries and define product boundaries.

4.2 CASE STUDY RESULTS

In this section, we present the results from our case study, uniquely through the lens of qualitative insights. Guided by the voices of our interviewees, we intricately weave their narratives into the fabric of our findings. We arrange these quotations into clusters that encapsulate shared perspectives, experiences, and underlying trends. This approach not only grants a platform for the participants' voices but also synthesizes their contributions into coherent themes. As we navigate through this section, these curated categories stand as windows into perspectives, allowing us to discern patterns, connections, and novel insights that collectively enrich our understanding of the research landscape.

4.2.1 Survey

We got 32 answers from people in Germany and Spain in the survey. Figure 8 shows the distribution by country. The answers from Spain are the majority, representing around 84%, and Germany represents about 16%.

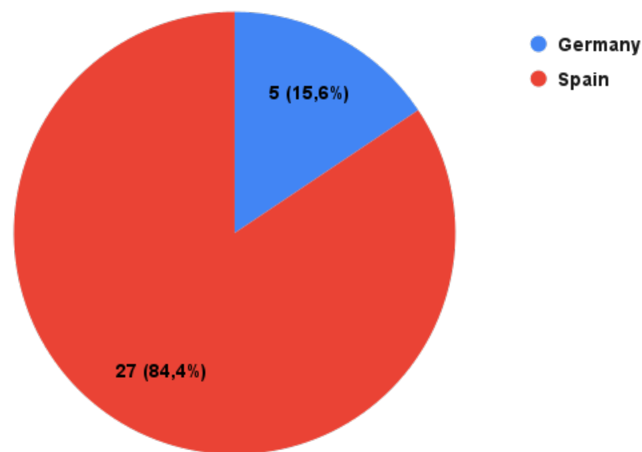

Figure 8 – Survey - Answers by Country

Figure 9 shows the interviewees' roles, where we can see that almost half of the respondents are Software Developers, followed by Team Lead (or Technical Lead) and Software Architect and Quality Analyst, in this order. In this question [SQ19], we also had the option to the respondent inform a different role, and we got SDET (Software Developer Engineer in Test) and IT manager as custom answers using the "Others" [SQ19A05] option.

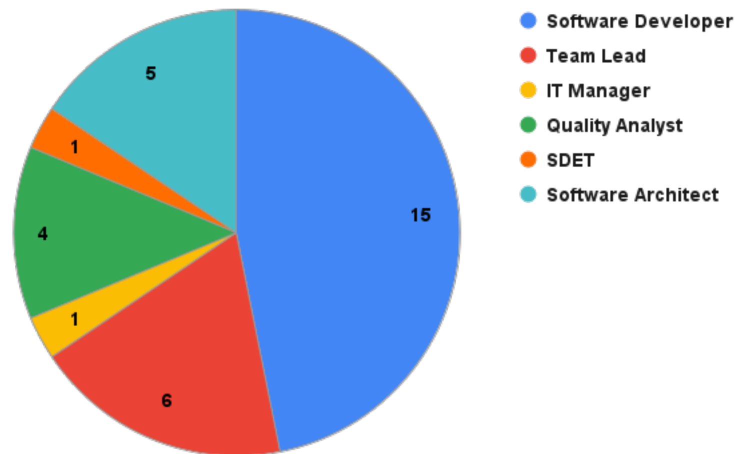

Figure 9 – Survey - Answers by Role

To understand the level of cultural diversity, we also asked the respondents [SQ22] to tell us how many people from different countries they had on their team. Figure 10 shows that the teams have people from multiple countries. Over 78% of the respondents told us that their team has more than three people from a country that is not yours. Almost 22% told that they have 2 or 3 people from a country that is not yours. The respondents also had the alternatives "Yes, 1 person" [SQ22A01] and "no" [SQ22A04].

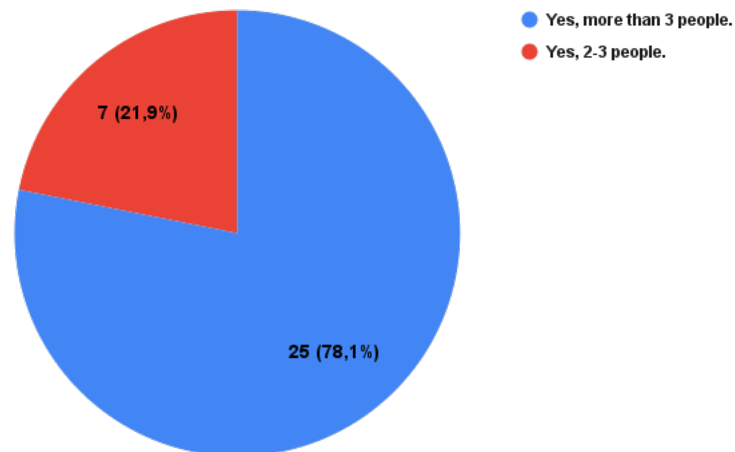

Figure 10 – Survey - Cultural Diversity

To identify our interviewees' background and market experience, we asked them how much experience they have working with software development. Figure 11 presents the consolidated answers. We identify that over 2/3 of our interviewees have more than five years of experience, representing 48,5% of the people with over ten years of experience and 29% of people with

experience between 5 and 10 years. We also have around 19% with between 3 and 5 years and experience and less than 5% with less than one year of experience.

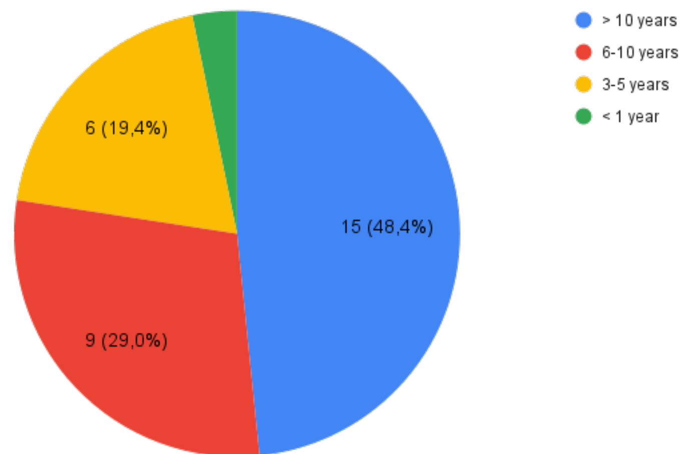

Figure 11 – Survey - Years of experience

In the first section of our survey, we asked about agile practices and the frequency that these practices were adopted. Figure 12 presents the answers from our interviews. Regarding automated testing and coding standards, it's also possible to see the standard deviation from each answer on Table 5, over 50% of our respondents answered that they always use it, and no one does not use it. When asked about collective code ownership, over 75% of the respondents said they adopt it often or always. When considering continuous integration, everyone said that they adopt it often or always, being 90% continually adopting and 10% often adopting. Pair programming had more heterogeneous answers, with around 28% of the answers that always or often adopt it and around 28% that never or rarely adopting it. Regarding refactoring practice, almost 75% of our respondents said they always or often adopt it. When asked about the requirements workshop, around 47% said they always or often adopt it. About Scrum of Scrum, 43% said they often or always use it. About simple/incremental design, 50% of our respondents said they always or often use it. When asked about Sprint demo/review, around 94% of our respondents said they always or often adopt it. Regarding Test-Driven Development (TDD), just 25% of our respondents adopt it regularly. Moreover, 91% of our respondents said they regularly adopt user stories.

	Practice	Standard Deviation
1	Automated Testing	0.98
2	Coding Standards	0.76
3	Collective Code Ownership	0.92
4	Continuous Integration	0.30
5	Pair Programming	1.16
6	Refactoring	0.73
7	Requirements Workshop	0.95
8	Scrum of Scrum	1.24
9	Simple/Incremental Design	1.01
10	Sprint review/demo	0.72
11	Test Driven Development	1.16
12	User stories	0.67

Table 5 – Standard Deviation - SQ1

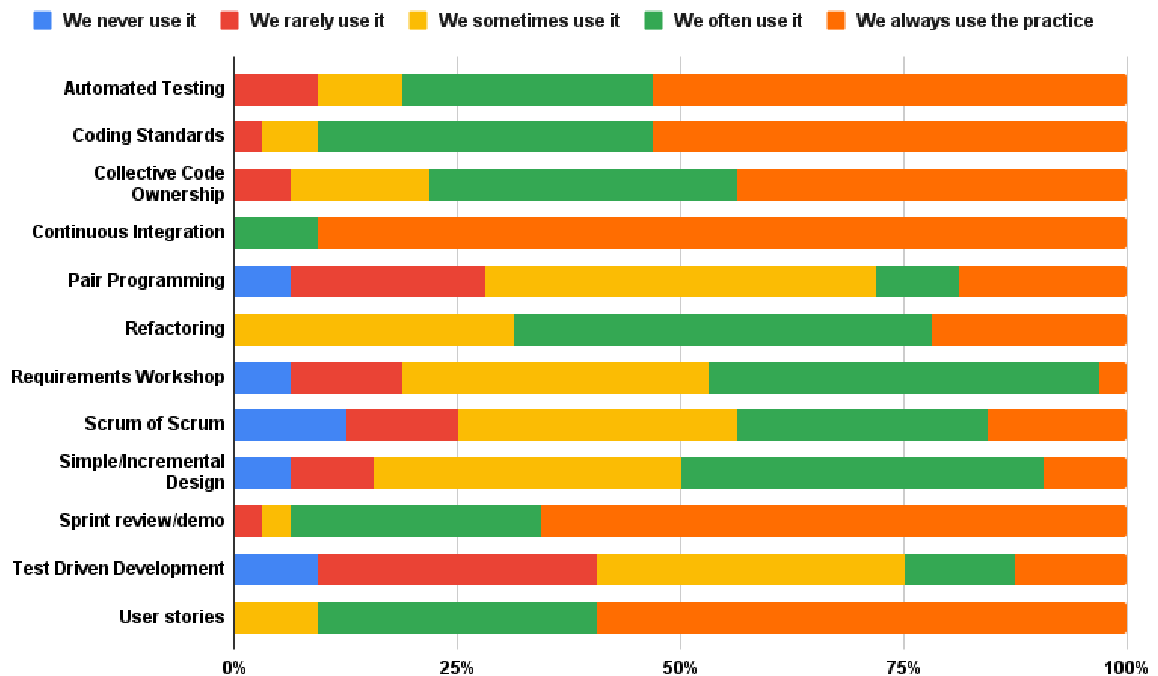

Figure 12 – Survey - Practices Adoption

Figure 13 presents the answer consolidation related to question SQ02 as described in Appendix C, it's also possible to see the standard deviation from each answer on Table 6. Regarding Application Programming Interface (API), 81% of our respondents said they always or often use it. When asked about Domain-Driven design, 62,5% said they always or often use

it. Event-Driven Design, 78% said they often or always use it. Regarding microservices, all the respondents said they always, often, or sometimes use them. When asked about Model-Driven Design, around 69% said they always or often use it. Regarding REST/RESTful practice, all the respondents said they often or always use it. When asked about Service-Oriented Architecture and Component-Based Architecture, 50% and 56% said they often or always use it.

	Practice	Standard Deviation
1	Application Programming Interface	1.02
2	Domain-Driven Design	0.73
3	Event-Driven Architecture	0.97
4	Microservices	0.67
5	Model-Driven Design	0.91
6	REST/RESTful	0.46
7	Service-Oriented Architecture	1.22
8	Component-based architecture	1.19

Table 6 – Standard Deviation - SQ2

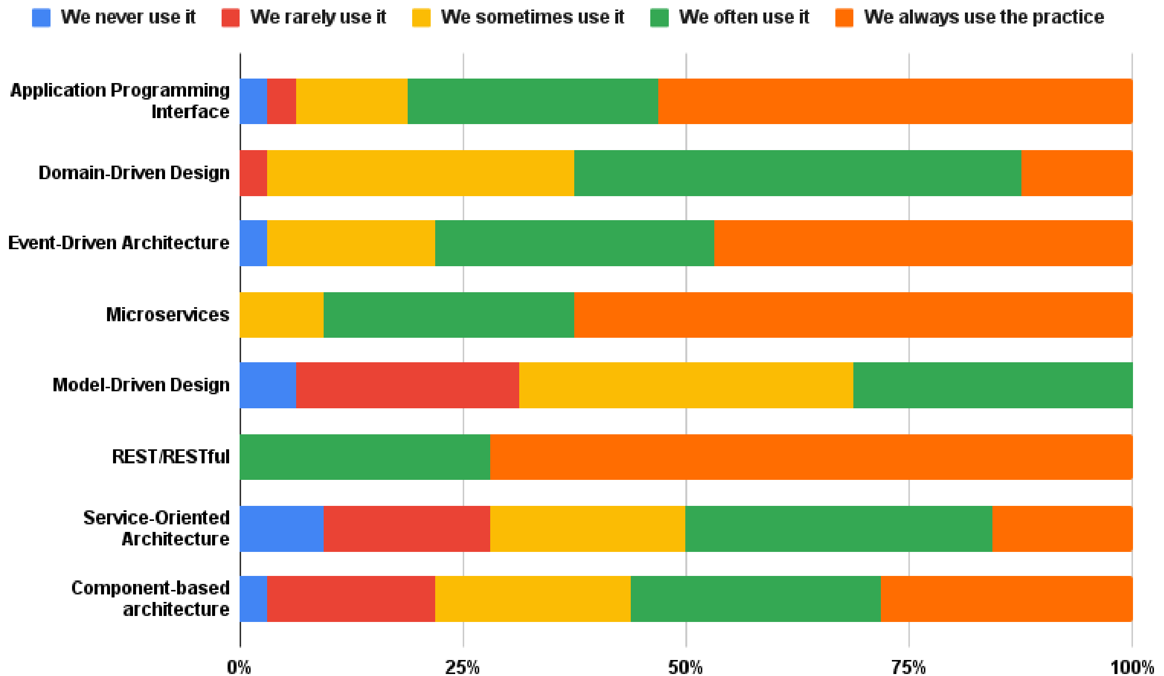

Figure 13 – Survey - Architectural rules on distributed teams

Figure 14 presents the answer consolidation related to question SQ03 as described in Appendix C, it's also possible to see the standard deviation from each answer on Table 7.

When asked if a lack of skills negatively impacts team communication, around 53% said they agree or strongly agree. When asked about lack of trust, around 85% of the interviewees said they agree or strongly agree that it negatively impacts team communication. Regarding the impact of language limitation on team communication, about 78% said they agree or intensely that it has a harmful impact. Around 62% said they agree or strongly agree that not being aware of cultural differences damages team communication. According to 87% of our interviewees, poor communication negatively impacts team communication. 66% of our respondents said poor documentation is bad for team communication. Moreover, 44% said that rare face-to-face interaction is unsuitable for team communication.

	Practice	Standard Deviation
1	Lack of skills	1.18
2	Lack of trust	1.18
3	Language limitation	1.10
4	Not being aware of cultural differences	0.98
5	Poor communication	0.98
6	Poor documentation	1.09
7	Rare face-to-face interaction	0.90

Table 7 – Standard Deviation - SQ3

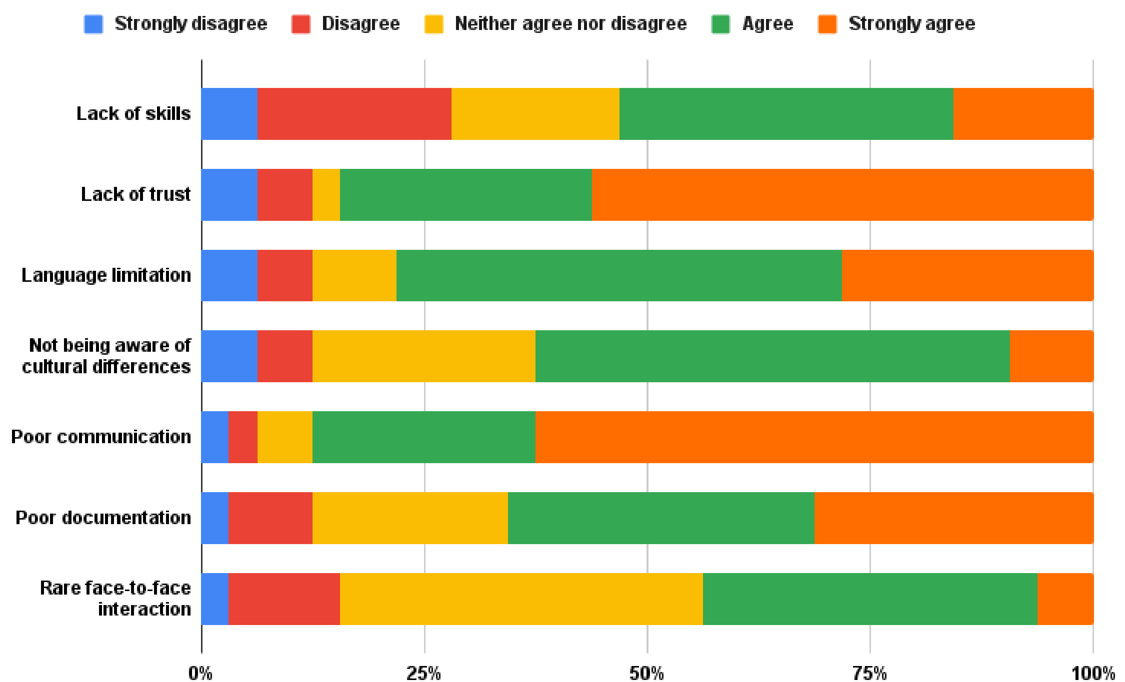

Figure 14 – Survey - Aspect impact over communication

Figure 15 presents the answer consolidation related to questions SQ04 to SQ17 as described in Appendix C, it's also possible to see the standard deviation from each answer on Table 8. In question SQ04, 81% of our respondents said they partially or agreed with the statement. In SQ05, 56% of our respondents said they partially or agreed with the statement. In question SQ06, around 40% of our respondents said they partially or agreed with the statement. When presented with the statement in SQ07, 47% of our respondents said they partially or agreed with it. In question, SQ08, around 44% of our respondents partially agreed or agreed with the statement. When presented with the SQ09 statement, around 38% of our interviewees said they partially agreed or agreed. In the statement, SQ10, around 69% said they partially agreed or agreed with it. In question SQ11, 56% of our respondents said they partially agreed or agreed with the declaration. In question SQ12, 87% of our respondents said they partially agreed or agreed with our affirmation. When asked the respondents' opinion about question SQ13, around 69% said they partially agreed or agreed. In question SQ14, almost every respondent said they partially agreed or agreed with the statement, representing around 94% and 6% answered they did not know how to answer. When presenting question SQ15, around 84% said they partially agreed or agreed with the affirmation. Regarding question, SQ16, 43% of our interviewees said they partially agreed or agreed with the statement. Moreover, 84% of our interviewees said they partially agreed or agreed with the statement presented in SQ17, and just 16% said they did not know how to answer.

	Question	Standard Deviation
1	SQ04	0.95
2	SQ05	0.91
3	SQ06	1.04
4	SQ07	0.72
5	SQ08	0.88
6	SQ09	1.04
7	SQ10	0.99
8	SQ11	1.01
9	SQ12	0.59
10	SQ13	1.02
11	SQ14	0.61
12	SQ15	0.88
13	SQ16	0.97
14	SQ17	0.69

Table 8 – Standard Deviation - SQ4 to SQ17

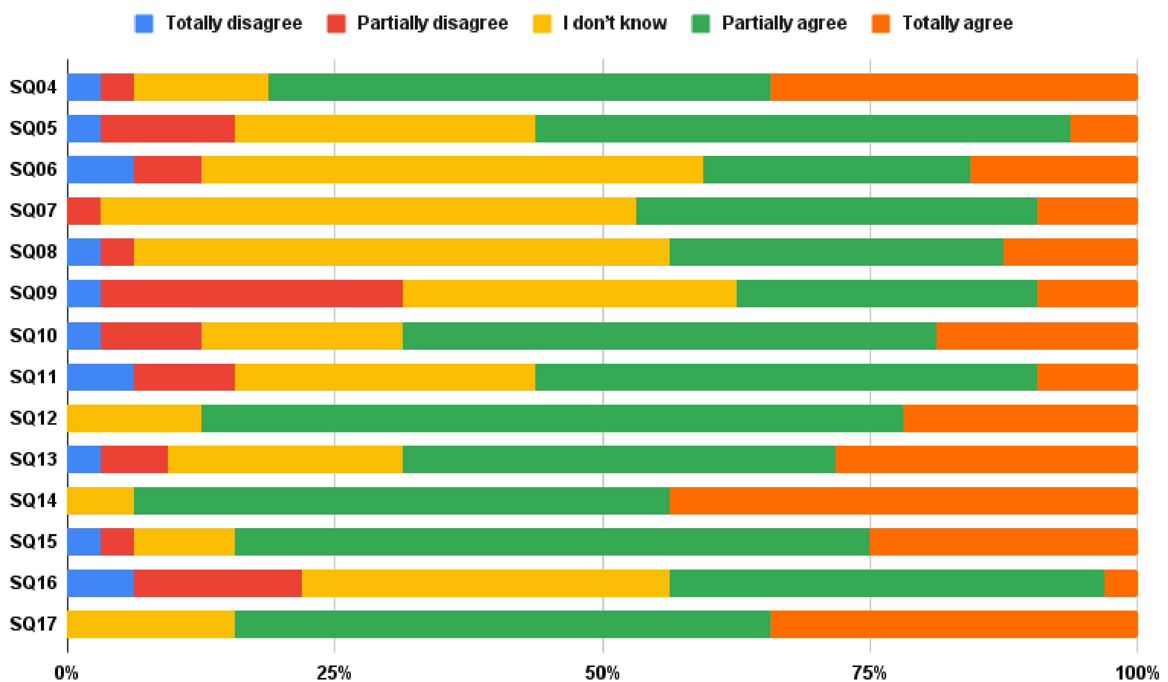

Figure 15 – Survey - Software Architecture over distributed teams communication

4.2.2 Interviews

In this section, we will present the results obtained from the interviews. When referencing the respondents, we will adopt the codes defined in Table 3 to identify them.

Based on the respondents' statements, we identified some practices related to software architecture (CAT1) that can enable communication: (CAT1.1) standards definition, (CAT1.2) adopting archetypes, (CAT1.3) decoupled architectures, (CAT1.4) teams independency, and (CAT1.5) guidelines.

Regarding the standard definition (CAT1.1), adopting it since the beginning of the project, along with the definition of which best practices the team should follow, can help to reduce misunderstandings (see quote EB-3 and EC-8) and create a base knowledge (see quote EC-10). Furthermore, adopting these standards can make adding them to a project easier (see quote EC-9). However, even with these standards and best practices described, they will still have their own opinion but with a defined base direction (see quote EB-1). The consequence of not having any standards is getting uncoordinated teams where everyone creates their standards (see quote EC-12).

- EB-3: *"Having common standards and best practices will not avoid misunderstandings. It will reduce it from the beginning, at least because everybody understands the same things about how to present solutions."*
- EB-1: *"But then everybody will have a different opinion. The important thing is that everybody shares the fundamental approach to do things, so when building a solution, everybody thinks about the same things, performance, reliability, all those things."*
- EC-8: *"As much standard is the technologies, you can avoid miscommunication because you are using standard globally familiar."*
- EC-9: *"Patterns which define the interface, the communication channel. The more defined the standards, and the easier it will be to add them to the projects."*
- EC-10: *"At the organization level, depending on the type of application that you have to build, should have architectural standards. For instance, creating a web application with a backend that will use Java. Establishing good practices, archetypes, or anything else that favors the creation process in any team, they will know what to do."*
- EC-12: *"Twelve development teams where everyone does what they want without any processed defined, every one defined its architecture."*

Although adopting practices (CAT1.1) can help to mitigate communication problems, just having it by itself will not avoid misunderstandings (see quote EB-10); sometimes, it needs to be complemented by other strategies like documentation (see quote EB-11). However, documenting every piece of information is not viable. It can be considered a bad practice, but document the basic information related to requirements, solution specification, and how the solution is built (see quote EB-12).

- EB-10: *"So you need the rules and the standards and the approaches common for everybody. But even having that, you could get to different solutions, and you can maybe misunderstand requirements."*
- EB-11: *"You follow your practices, your rules, your tools, your templates, and on top of that, you document things."*

- EB-12: *"So it always helps to document everything bad, not doing too much documentation, not saying this, but the basic things, right, the requirements, the solution, and the why did you come up with that with the solution."*

In addition, participants mentioned that adopting archetypes (CAT1.2) helps to avoid misunderstanding (see quote ED-1) and can improve the maintainability of systems because you are going to have base code generation (see quote ED-2), which helps the support team to maintain it.

- ED-1: *"I think these patterns help; for instance, we use maven archetypes."*
- ED-2: *"Today, we use archetypes to generate more than ten applications, consumers of Kafka messages. Imagine if we did not have this archetype to generate these applications, everyone would create their application with a different structure, and it would be more difficult for the support team to maintain it."*

Another practice mentioned was decoupled architectures (CAT1.3). This type of architecture can generate more independency between teams (CAT1.4) and become easier to manage these teams (see quote EC-1). Although this architecture can ease management, it can generate knowledge silos (see quote EA-1). Moreover, adopting decoupled architecture like microservices or event-driven can also bring common understanding because, in the end, they are standards used globally (see quote EC-6, EC-7, and EC-15). It also brings the benefits where the risks are isolated; if any part of an application stops working, the impact is negligible because the structure is decoupled (see quote EA-18). Even with these complexities and benefits, the concerns about communication problems will vary based on application complexity architecture-wise (see quotes EA-4 and EA-8).

- EA-1: *"Having distributed teams, especially if the architecture is, you know, or microservices architecture which suggests that you have a good level of decoupling. Furthermore, that sometimes creates silos, and this is where the communication or collaboration problem might occur."*
- EC-6: *"Use a microservice architecture where established the interfaces and the interactions between different microservices. The market will always use a standard architecture that is easily understandable."*

- EC-7: *"It does not matter if you are working from Brazil, China, or Argentina, everybody is going to know what it is a REST service or what it is an API REST or even when we have a messaging system, where receives events and react to them."*
- EC-15: *"We create the microservices with integration layer with other systems, using Kafka and API REST. If we have newcomers, we have defined standards and a clear direction."*
- EA-18: *"It really depends on the complexity, the overall complexity of the product they have to deal with. When it comes to architecture, if things are really decoupled in your architecture, be it event-driven architecture or just microservices architecture where just a lot of orchestration is happening, and people are calling your services. The impact of doing something work is smaller because the structure is decoupled."*
- EA-4: *"If you are still doing something relatively simple, right? But, at a point when you tackle something more serious, more complex, these challenges will start popping up with, you know, coming from the differences of, in the culture and expectations mainly."*
- EA-8: *"If the product itself is relatively simple, architecture-wise, in my opinion, the problem is not that big."*

Although some respondents believe that adopting microservice can improve communication because of team independence (see quote EC-1), it can also be problematic when you need these independent teams to communicate between them (see quote ED-3) and also consider that you have to manage different cultures (see quote EC-5).

- EC-1: *"Having loosely coupled architectures because it supports teams to work more independently. Becoming easier to manage these teams, does not matter where they are, because their work is more independent."*
- ED-3: *"So I believe that inside of a single service, the communication can have a good flow, but when different teams develop the microservices, and they need to communicate, it can be a problem."*
- EC-5: *"I believe that you will have to put more effort into coordinating the teams to keep the different teams aligned. Managing these differences will not always be easy when we have different cultures. Contrariwise, it is more difficult, and we must dedicate more time."*

Guidelines (CAT1.5) is another category identified during the analysis. Even more, it is essential to have some guidelines when the company is not an IT company (see quote EA-13) and can be used to coordinate between teams working on different products (see quotes EA-12 and EA-11).

- EA-12: *"I think having guidance architectural uh guidelines and um embracing that to the team or even on a higher level, if you are leading, let us say, a department, right? And you have multiple teams who are working on different products, not even connected products to each other."*
- EA-11: *"Where you have ten teams working on ten products, and they all are distributed. So having guidelines, the development guidelines and architectural guidelines is absolutely required to and in an ideal world that would be sufficient."*
- EA-13: *"Especially in the companies that are not, you know, pure IT companies, pure companies that are the main thing of which is to create the software as a service or sell software or services uh digital services companies."*

Communication Practices (CAT2) - Another group of categories identified based on respondents' statements was the communication practices communication practices where we have five sub-categories: (CAT2.1) communication facilitator, (CAT2.2) face-to-face meetings, (CAT2.3) less-interaction, (CAT2.4) agile ceremonies and management frameworks, and (CAT 2.5) alignment meetings

Regarding the communication facilitator (CAT2.1), having a person playing a senior role in the team, like an architect or a senior engineer, who works as a facilitator between the distributed teams, could help mitigate and keep the teams connected (see quote EA-2, EC-11, and EA-10).

- EA-2: *"I think the key role of architects and more senior engineers, in that case, is to ensure that the teams are staying connected and this high-level understanding of how things are connected."*
- EC-11: *"An architect at the organization level who manages the integration between the teams and the standards."*
- EA-10: *"If the team feels that they are not there and this collaboration gap exists, or communication gap exists, or creative ideas are not coming up or surfacing up, then I*

think it is a responsibility of senior staff, senior engineers, and maybe lead engineers, architects."

When talking about face-to-face meetings (CAT2.2), having at least a single meeting to know each other face-to-face can help to mitigate possible problems and build trust between team members (see quote ED-5), and the integration between team members becomes easier (see quote ED-6). Sometimes, the consequences of a lousy communication flow are blocked in a daily task (see quote ED-7). Moreover, a lack of communication or synchronization between team members can cause duplicate work (see quote EB-2).

- ED-5: *"If you know the person face-to-face, you will have a better communication flow and openness to talk."*
- ED-6: *"At least, I believe you will have open communication when the people know each other in person. The integration will become easier."*
- ED-7: *"Some problems communicating with another team, with private messages in Microsoft Teams or even emails, but did not get any answer. Ultimately, this communication problem blocked my task."*
- EB-2: *"important is to often meet with the people and, and always try to be the other what you are working on because sometimes, as you are in different locations, you might be working on similar problems and maybe you are two people building a similar solution for a similar problem."*

Regarding less interaction between teams (CAT2.3), it can allow fewer errors when adopting a decoupled architecture, and the teams work in a different workflow, which does not require constant communication (see quote EC-4). Although fewer errors can be achieved with these practices, they can also generate knowledge silos, as presented in independency between teams (CAT1.3).

- EC-4: *"Obviously, the interaction between different workflows will have less interaction. The less interaction allows fewer errors and fewer misunderstandings in general."*

Agile ceremonies and management frameworks (CAT2.4) can help spread knowledge and build a shared understanding between team members (see quote ED-4). Sometimes, adopting

these ceremonies is impossible because of team distribution and lack of timezone overlap (see quote EC-14).

- ED-4: *"Clarify the requirements, for instance, adopting refinement meetings, estimation and planning meetings, and OKR frameworks. Furthermore, preserve these meetings to maintain the requirements updated and clear to everyone."*
- EC-14: *"At some point, we had one hour to do every ceremony because of the timezone; it was the only common hour between the teams."*

Regarding alignment meetings (CAT2.5), it can be leveraged to mitigate communication problems. There are meetings like check-in meetings (see quote EB-4), feedback meetings (see quotes EB-13, EB-15), or even requirement refinement (see quote EB-14). Although having alignment meetings could help to improve communication, not having some direction in what practices must adopt and which standards to follow can drive complete confusion about where to establish their standards (see quote EC-13). Feedback is a crucial factor when talking about communication in distributed teams (see quotes EB-7 and EB-5); it also can be used as a knowledge-sharing mechanism (see quotes EB-16 and EB-9) and consequently help to build a reliable and open environment (see quotes EB-9 and EA-7).

- EB-4: *"If you have the team spread across different locations, it always helps us to get periodic check-ins with the teams."*
- EB-13: *"But then on top of that, you will have meetings to uh hear from all the other people."*
- EB-14: *"Whenever there is a requirement that comes from business, usually it, it comes to the business analyst, and then the business analyst sits with me in this case, and then I kind of decide whether this needs architectural design behind or not."*
- EB-15: *"Draft of the proposal. And then what I do is I go to the development team, and usually I invite the whole development team and not just develop the, develop the lead here. And I propose a solution."*
- EC-13: *"It does not matter if you have aligning meetings; in the end, everyone does what they want."*

- EB-5: *"Somebody comes up with a solution, and then you present it to the group of architects, and everybody gives feedback, and then this works in different locations."*
- EB-7: *"It is good to put everything written, right? Because, again, talking, you can have misunderstandings. But then if everybody, everything is written and everybody reads it and then comes back with feedback, then you will uh reduce these misunderstandings."*
- EB-16: *"I send it beforehand the meeting so they can have a look and come up with questions to the meeting. And then in this meeting, I do the presentation, and then I open for Q&A, and then everybody has ideas."*
- EB-9: *"So these are my the requirements I got, and people ask questions, I do not understand this, I do not understand this. And then maybe you need to get together with the owner of the requirements to have some sessions, some it relations until everybody is on the same page, right? And then once the -requirements are clear, then you go to the solution and then for the solution there."*
- EA-7: *"Be kind on listening what others have to say in terms of, what is not working, what can be done better, and so forth, and make it transparent for the team."*

Cultural aspects (CAT3) is another category identified while analyzing our interviewees' answers. It is going to be presented in three sub-categories: (CAT3.1) learning culture, (CAT3.2) transparency, and (CAT3.3) commitment.

Regarding learning culture (CAT3.1), it is complex to manage but necessary to spread knowledge when adopting complex architectures, even more, when the team has different seniority levels, so it is necessary to develop a learning culture (see quote EA-9). How each person handles and absolves knowledge will vary based on each background (see quote EA-26), some individuals do not have the necessity to understand the basics behind frameworks, architecture (see quote EA-20), or tools used in the development process (see quote EA-25) and to have a quality product the foundation needs to be comprehended (see quote EA-23). Moreover, the learning culture is essential to build a shared understanding (see quote EA-19), and one of the challenging parts is to create this culture in the newcomers (see quote EA-6) apart from the cases where the architecture complexity becomes challenging to keep up the level of understanding (see quote EA-20).

- EA-9: *"If the topics are complex and you have different teams with different cultures, with different seniority levels, this is, of course, affects overall, let us say, effectiveness and performance, and to tackle that, I think it is about, it is about creating a proper learning culture in the team."*

- EA-26: *"In different countries, it might be different, depending on the university, depending on the environment."*

- EA-25: *"High-level abstractions more and more, and they are being used in the modern architectures because we try to be as efficient as possible, bringing in all those cool frameworks that are out there that are hiding a lot of complexity and, and modern uh modern young generation of engineers. They do not bother themselves to really understand how certain things are done and how certain patterns are being implemented. And this is some, especially depending on the culture as well."*

- EA-23: *"It always starts from really basics in the foundation. It is like humans are trying to learn how to walk first, and then they do the rest."*

- EA-20: *"Architecture is just more complex topic because, you know, for some people, very distributed systems with a lot of services and a lot of teams involved. It might be really challenging to keep this high-level understanding in the head."*

- EA-19: *"In order to mitigate communication problems again, when it comes to architecture, how the product is done is to make sure that everybody who is participating in that collaboration and communication is ramped up, they understand the basics."*

- EA-6: *"Everybody who is coming to that team coming originally from totally different background, they have to buy it in, and that is the difficult part, architecture."*

Transparency (CAT3.2) is also related to two other essential points: knowledge management and openness culture. To overcome the challenges generated by distributed teams, the team needs to have a way to be fully transparent about the reason for doing something (see quotes EB-8, EB-6, and EA-22) and have this information flowing; it is essential to keep every person in the same understanding level (see quotes EA-21 and EA-24).

- EB-8: *"There is no way to remove them completely, but you will really reduce them if you have all the decisions, documented the requirements, clearly documented, and you share these documents."*

- EA-24: *"Full transparency of why it is like that and um and the culture of being open for, for listening back uh what can be improved. It is universal, to all architectures, maybe with some small differences. But in essence, it does not really matter what kind of architecture you have."*

- EA-21: *"Another problem is why architectures, in general, might be challenging for distributed teams to communicate simply because insufficient information flows effectively. But that also means that I feel like putting myself in the shoes of an engineer and software developer programmer is that before jumping on the task, that is, to implement a certain thing in whatever the ecosystem is being highly distributed, event-driven, whatever is that I understand what I am doing."*

- EB-6: *"Record these decisions in, for instance, architectural records where you can say for this problem, we have decided this and this and this and these, the rationale behind these answers. And with that, you bring clarity for everybody, and also you document your decisions."*

- EA-22: *"challenge definitely will be their silos not enough information flowing, but I think in order to overcome it, we need to again, work on basics, works on foundation, mature the teams a little bit, make sure engineers are comfortable asking questions and then it is up to whoever is sharing information and whoever is receiving to ensure that that this information that is being shared is enough."*

Commitment (CAT3.3) is also considered by the respondents because even though the guidelines are defined, it does not mean that they will adopt them. Commitment is necessary for team members to adopt these guidelines (see quote EA-14).

- EA-14: *"If those architectural guidelines, are defined, it does not mean they will be used. It is really difficult to embrace them because everyone has their own opinion and can justify things that we cannot do that because we do not have time or skills or whatever and practice. I think it requires a bit more than just having architectural guidelines."*

Although some respondents may say that architecture can be a critical factor in a distributed environment, others say that, in some situations, it is better to give more attention to building trust over defining architectural principles (see quotes EA-5, EA-16, and EA-17) and understanding the capabilities inside the team (see quote EA-15).

- EA-5: *"I think it is not that much about, you know, architectural principles. It is exactly about building trust, um repeating things uh uh If necessary, making sure that everybody is on the same page, um double checking that time to time."*
- EA-15: *"The first thing to start is to assess the team and understand the skills."*
- EA-16: *"Practices we are using when it comes to development practice, maybe we are really struggling, predicting how much time certain things will take because we are showing we are not trusting each other and then, can be the thing um because it reduces the pressure of committing for a certain scope for two weeks, then we see how things are done."*
- EA-17: *"When the team believes that it is ok, we gain a little bit of knowledge and trust for each other. We can switch to Scrum and make sure that as a team, we can commit to a scope, same with architectural principles."*

4.3 CLOSING REMARKS

In summary, this chapter's results underscore the potency of the integrated approach that melds systematic mapping with a case study. The systematic mapping unveiled the broader landscape of existing research, elucidating prevailing trends and revealing gaps in our understanding. The subsequent case study, enriched by interviews and a comprehensive survey, offered a focused exploration of a specific context, breathing life into the quantitative insights with qualitative narratives. These methodologies not only validated our findings but also afforded a holistic perspective that transcends the limitations of individual methods. As we move forward, the insights gleaned from this chapter not only contribute to the academic discourse but also offer practical implications for our research domain. We continue to discuss the results in the following chapter.

5 DISCUSSION

As we focus on the discussion chapter, we step into the heart of our research journey, poised to interpret the multi-dimensional insights we've amassed. Drawing from the rich wellspring of systematic mapping and case study results (Chapter 4), we explore our research domain's contours comprehensively. This chapter serves as an intellectual crucible where the empirical findings merge with theoretical underpinnings, offering a nuanced perspective on the themes and patterns that have come to light. We scrutinize the implications of our systematic mapping's revealed trends through a critical lens, identify gaps that warrant further exploration, and delve into the resonant narratives from the case study's interviews and survey responses. In addition to this analysis, the chapter also serves as a repository of lessons learned — a testament to the methodological synergy harnessed, the challenges surmounted, and the garnered throughout this scholarly. As we navigate through the pages of this chapter, we synthesize our findings with academic discourse, spotlighting novel insights, potential avenues for future research, and the broader implications our study extends to theory and practice.

5.1 SYSTEMATIC MAPPING

In the following, we will discuss our study's results, highlight the main findings, and relate them to our research questions. The outcomes of this section will be hypotheses extracted based on the results, which will drive our subsequent studies.

5.1.1 How software architecture design impacts the DSD environment?

Alzoubi and Gill [SM13] bring to our attention that the Agile Enterprise Architecture (AEA) (HERBSLEB; MOITRA, 2015) uses a standard information model that can enable clear communication in distributed teams. This model can generate a common language between the development groups and improve communication because system and software structure definitions are needed. Adopting a component-based strategy to build an application using component interfaces or contracts can enhance communication by reducing communication overhead.

Although adopting modular architecture may reduce communication overhead and lessen

misunderstanding during the development process, some authors point out that this approach can generate other challenges related to poor communication and sometimes provokes team isolation (BANO; ZOWGHI; SARKISSIAN, 2016). Moreover, a mature architecture is essential to simplify transparent task distribution (NOLL; BEECHAM; RICHARDSON, 2011).

Sievi-Korte [SM10] highlights that API is a crucial architecting practice to define product boundaries and handle modules' limits. Component-based and APIs (JACOBSON; BRAIL; WOODS, 2012) strategies are examples of interface-driven design. They also follow less type interdependencies and loosely coupled principles, which, according to Sauer [SM09], is a communication enabler on distributed projects. Tekinerdogan et al. [SM08] also point out that the loosely coupled principle is the most acceptable practice when architecting software.

Still, about the impact of software architecture and communication in DSD teams, Mishra and Mishra [SM07] and van Vliet (VLIET, 2008) affirms that software architecture can be used to reduce the need for communication in a multi-site development project. Moreover, it is possible to use the architectural structure of the system to split work between sites, which indicates a variation of Conway's law (CONWAY, 1968).

Microservices is an example of an architectural style that applies both type interdependency and loosely coupled tenets. It has become widely adopted (FRANCESCO; LAGO; MALAVOLTA, 2018), and some authors consider it reasonable to follow Conway's law (BALALAIE; HEYDARNOORI; JAMSHIDI, 2014). Lenarduzzi and Sievi-Korte [SM06] bring to our attention that adopting microservices has some pitfalls, even more related to communication, making the teams rely on software architects to coordinate. However, having this coordinator role in an environment has pros, like a single point of contact to manage problems, and cons, like decreasing visibility or making the team non-democratic.

Crnkovic (CRNKOVIC, 2001) define software component as: "*[...] a composition unit with only a contractually specified interface and explicit context dependencies. A software component can be deployed independently and is subject to composition by third parts.*"

But what is the relationship between software component independency level (or coupling level) and software design quality? Page-jones (PAGE-JONES, 1988) affirms that: "*The first way of measuring design quality [...] is coupling, the degree of interdependence between two modules. Our objective is to minimize coupling; that is, to make modules as independent as possible.*" This affirmation drives us to believe that the software architecture quality is a consequence of how the software modules communicate between them. Meanwhile, Sauer [SM09] and Bosch and Bosch-Sijtsema (BOSCH; BOSCH-SIJTSEMA, 2010) observe that adopting

loosely-coupled components design is critical to mitigate communication problems in DSD environments.

These findings suggest that adopting a loosely coupled software design strategy can mitigate communication challenges in DSD environments. Based on this, we built our first supposition:

SP1 Decoupled software architectures are communication enablers and can help to mitigate communication challenges in DSD environments.

5.1.2 Is there any architectural design that can positively impact the DSD environment?

According to Alzoubi and Gill [SM13], the Agile Enterprise Architecture (AEA) can be used as an integrated shared view to help achieve the best design and architecture. Ovaska et al. (OVASKA; ROSSI; MARTTIIN, 2003) also support using AEA as an integrated shared view. When talking about performance on DSD teams, Alzoubi and Gill [SM13] show the contrast between adopting AEA and using EA, where no value is delivered, which the AEA definition supports (EDWARDS, 2006). However, Alzoubi et al. [SM12] indicate how AEA affects DSD teams still needs clarification. Although AEA's impact is unclear, Kornstädt and Sauer [SM05] and some authors (FARIA; ADLER, 2006; HERBSLEB; GRINTER, 1999) highlight adopting architecture-based development brings advantages to the DSD environment, including the coordination and task allocation.

Urrego et al.[SM01] and Kornstädt and Sauer[SM04] bring concerns about the distances between teams and significant communication gaps, which sooner or later will lead to miscommunication, causing a lack of trust and issues related to issuing the resolution. These problems bring even more apprehension, mainly to projects with complex applications. To mitigate these communication challenges, Kornstädt and Sauer [SM04] and Gill and Alzoubi [SM11] recommend adopting practices that enable daily communication between team members and feedback loops. These recommendations indicate that following some agile rules, for instance, from Extreme Programming (BECK, 2001), Scrum (SCHWABER; SUTHERLAND, 2011), and DevOps (EBERT et al., 2016), may help to mitigate communication challenges which some authors also support (HOLMSTRÖM et al., 2006; JAIN; SUMAN, 2016; VALLON et al., 2018).

We found two perspectives when discussing architectural knowledge and knowledge man-

agement in global software development. The first one is the impact of architectural expertise on the performance of distributed teams. Clerc [SM02] indicates that the teams use the obtained knowledge regarding architecture knowledge to make decisions, but distributed teams do not accept it, which is supported by some authors (ALI; BEECHAM; MISTRIK, 2010; CLERC; LAGO; VLIET, 2009). The second one is knowledge management in general and its impact on communication challenges. Clerc et al. [SM03] indicate different practical strategies to spread knowledge through an organization and present two use cases where these practices had a real impact.

These findings suggest adopting agile practices combined with an architectural design focused on architecture as a decision-making process in DSD environments can help to mitigate communication challenges. Based on this, we built our second supposition:

SP2 An Architectural Design which enables agile practices and follows architectural-centric principles can help to coordinate DSD teams and mitigate communication challenges

5.2 CASE STUDY

In this section, we will discuss the results obtained from our case study and compare them with the outcomes from the first step of this work, the systematic literature mapping. To facilitate understanding, the findings of this work are categorized into three different perspectives, architecture-wise, communication, and cultural aspects. The following sections present each one of these perspectives.

5.2.1 Architecture

During our investigation, we identified aspects of software architecture from different perspectives, including coding standards, adopting archetypes, decoupled architectures' impact on communication and team organization, team independency, and development guidelines.

5.2.1.1 *Coding standards*

The interviewees point out that having coding standards will reduce misunderstanding from the beginning of the project (see quote EB-3), and it can also avoid miscommunication by adopting standards known globally (see quote EC-8). ISO/IEC/IEEE 12207:2017 (ISO/IEC/IEEE..., 2017) considers coding standards to be guidelines which can mean that following them can avoid other coding quality issues and improve maintainability and readability. Holmström et al. (HOLMSTRÖM et al., 2006) shows that large companies like Intel and HP have adopted coding standards at the organization level combined with agile practices. Moreover, Kircher et al. (KIRCHER et al., 2001) show that applying coding standards does not require a co-located team. The SOLID principles can be considered coding standards and a recommended practice to mitigate communication problems [SM12].

Multiple patterns can be adopted as coding standards, and having them defined, depending on the level of detail it has, will be easier to add them to projects (see quote EC-9); having this base understanding of what is fundamental to the project and which knowledges are necessary to build a solution is essential in the development process (see quote EB-1). Of course, the standards will depend on the type of application the team is building (see quote EC-10). The development process can become distressful if no method is defined and each team starts to create their approach (see quote EC-12).

Moreover, in SQ01 at SQ01A02, of the survey, where we asked about adopting coding standards, around 90% of our respondents said they always use or often use it, with no one declaring that they never use it. These answers had a standard deviation of 0.76 which means that the answers were close to the mean answer in this case.

5.2.1.2 *Adopting archetypes*

During the interviews, interviewees came up with a concept not raised during our systematic mapping, where adopting it could create a common starting point for every team. Following archetypes, like maven archetype (MA; LIU, 2020; VARANASI et al., 2014), would help to mitigate communication problems and divergencies when following organization standards (see quotes ED-1 and ED-2).

5.2.1.3 Decoupled Architecture

During the interviews, respondents indicated that decoupled architectures might generate more independence between teams (see quote EC-1), reducing the necessity for inter-team coordination [SM07, SM13]. Furthermore, decoupling architectures are suitable to mitigate risks; if any part of the application stops working, its impact is limited because of the structure decoupling (see quote EA-18).

Another benefit of adopting this type of architecture is that it is widely known and accepted by different organizations. This can ease the understanding process and establish a base knowledge between team members (see quotes EC-6, EC-7, and EC-15). It is also essential to consider that decoupled components constitute the most acceptable practices of software architecture strategy [SM08], or even more, adopting an API that can help to define modules and product boundaries [SM10], following a more separate structure of components is more likely that organization will be to develop them successfully on multiple sites [SM09] and therefore following Conway's law (CONWAY, 1968). Although there are many benefits from applying decoupled architectures, there are also some pitfalls, for instance, whether the structure is decoupling sufficiently or not (SIEVI-KORTE; RICHARDSON; BEECHAM, 2019) and may originate knowledge silos (see quote EA-1).

Some interviews said it is unnecessary to establish architectural practices when the system has a less complex structure architecture-wise; otherwise, tackling it may be necessary (see quotes EA-4 and EA-8). Furthermore, in complex structures like microservices, it becomes essential to have a good alignment and communication flowing between teams (see quote EA-3), and it may be necessary to have someone coordinating this communication (see quote EC-5).

Moreover, during our survey, we asked the respondents what they thought about the software architecture's impact on distributed teams. In our SQ04, we asked them if they believe software architecture can positively impact software development teams in a distributed environment, where 81% said they agree or strongly agree with the statement. It only indicates that architecture may impact communication from our interviewees' perspective, yet we need to clarify which type of architecture.

In SQ05, we asked if they believed that adopting microservices or micro-frontends can help improve communication between team members in multiple locations; 56% of our respondents agreed or strongly agreed, 16% disagreed at some level, and the other 28% were considered not

willing to give their opinion or did not know how to answer. These answers may indicate that one type of architecture that can improve communication would be decoupled architectures, like microservices or micro frontends.

In SQ06, we kept asking their perspective about other architectural patterns that may also impact communication in software development, and we asked: “Event-driven architecture can help improve communication between team members in multiple locations”; around 40% of our respondents said they agreed or partially agreed with it. Event-driven architecture is another architecture that helps improve communication in distributed software development environments. In SQ07, we asked the interviewees’ opinion: "Component-based can help improve communication between team members in various places." in this case, 47% said they agreed or partially agreed with it. In these two questions, around 50% of the respondents were unwilling to give their opinions or did not know how to answer. Both approaches are considered decoupled architecture and may help improve communication in a distributed environment. However, clarifications are necessary to understand the pitfalls of adopting these architectures and whether respondents agreeing with it could have been higher.

In SQ08, the statement has more relation with a philosophy that may impact the architecture development rather than architecture itself. The affirmation is: “Domain-Driven Design can help improve the communication between team members on numerous sites.”; 44% of the respondents said they agreed or partially agreed with this statement, while 50% did not know how to answer or were unwilling to give their opinion, and 6% disagreed or partially disagreed. It may indicate that the interviewees think philosophies can also contribute to mitigating communication problems in distributed software development.

Although the findings from SQ06 and SQ07 may indicate that software architecture adopting a decoupled structure may help to mitigate communication problems, in SQ09, the statement was “a decoupled component is a solution to mitigate communicate challenges”; and the answers were balanced, 38% agreed totally or partially, while 31% did not know how to answers or were unwilling to give their opinion and 31% disagree totally or partially. These answers do not help to make any conclusions about the statement related to decoupled components.

In SQ10 and SQ11, the objective was to understand the respondents’ perspectives on decoupled components. The SQ10 and SQ11 were “Microservices generate decoupled components” and “Having decoupled components architecture reduces the necessity for inter-team communication.”; and their answers in SQ10 were 69% agreed totally or partially, while 19% did not know how to answer or were unwilling to give their opinion, and 12% disagreed to-

tally or partially; their answers in SQ11 were 56% agreed totally or partially, while 28% did not know how to answer or were unwilling to give their opinion and 16% disagreed totally or partially. It may indicate that microservices architecture reduces the necessity for inter-team communication.

In questions SQ13 and SQ15, the objective was to get the respondents' point of view about microservices and APIs and their relation with each other. In SQ13, the statement was, "Microservices can provide a better component overview and enable less coupling between applications."; and the answers were 69% agreed partially or totally, while 22% were unwilling to give their opinion or did not know how to answer, and 9% partially or totally disagreed. In SQ15, the statement was, "Using APIs generates less interdependency between components and enables microservices."; and the answers were 84% agreed partially or totally, while 10% were unwilling to give their opinion or did not know how to answer, and 6% disagreed partially or totally. It may indicate that combining these practices, microservices and APIs, helps create a less-dependent environment and a better structure overview.

5.2.1.4 Team independency

Regarding team independence, our findings indicate that it is connected with the coupling level adopted by the software architecture being built. Having loosely coupled architectures supports teams in working more independently, making it easier to manage these teams regardless of location (see quote EC-1). Although it has some drawbacks [SM06]; for instance, it may generate knowledge silos, and more challenging to have effective coordination between teams (see quotes EC-3 and EC-5).

5.2.1.5 Guidelines

During our study case, another practice that came up was the adoption of guidelines during the software development process. Guidelines are essential at the department or company level (see quote EA-12), especially when it is not a pure IT company (see quote EA-13). These guidelines can be architecture-wise or development and are required in an ideal world (see quote EA-11). Having commonly acknowledged guidelines also affects modularity and change management, ultimately impacting the design decision process (SIEVI-KORTE; BEECHAM; RICHARDSON, 2019).

5.2.2 Communication

Other aspects discovered during this research were related to communication. In the following sections, we are going to discuss five categories related to communication: communication facilitator, face-to-face meetings, less interaction, and agile ceremonies and management frameworks.

5.2.2.1 Communication Facilitator

A communication facilitator is essential to keep teams connected and a high level of understanding; Someone with a good seniority level, like an engineer or architect, can perform this role (see quote EA-2), avoid communication gaps, and keep creativity at a reasonable level its also on the scope of their responsibility (see quote EA-10). The architect at the organizational level manages team integration and defines common standards (see quote EC-11).

There are multiple communication structures that an organization can adopt; nevertheless, these structures have pros and cons. Suppose the structure adopted is a one-level hierarchy, where one person manages the problems and will be the only one responsible for those decisions. In that case, it may create a non-democratic environment and requires a good level of expertise by the manager. There is an intermediate structure called a two-level hierarchy, where the approach is to follow a decision-making chain, and every microservice has a coordinator; however, it may generate synchronization problems in the communication between coordinators. An environment with a complete democratic focus is also possible, decreasing the possibility of team members' exclusion in the decision process, but the discussion may take longer. [SM06].

Sometimes, having a developer perform the facilitator role is also possible (RÄTY et al., 2013). Although, in most cases, a person performs the facilitator role, adopting an ontology can also establish a common vocabulary and help avoid communication misunderstandings (ARANDA; VIZCAÍNO; PIATTINI, 2010).

5.2.2.2 Face-to-face meetings

Face-to-face meetings are only sometimes possible in distributed environments because geographical distance decreases these opportunities [SM11]; the lack of face-to-face meetings may affect communication among distributed team members or cause misunderstandings in design (KHAN; BASRI; DOMINC, 2014; JAN et al., 2016).

Having a single meeting to know each other face-to-face can help to mitigate possible problems and build trust between team members (see quote ED-5), and the integration between team members becomes easier (see quote ED-6). Sometimes, the consequences of the lousy communication flow are blocked in daily tasks, or a lack of communication or synchronization between teams may cause duplicated work (see quotes ED-7 and EB-2).

During our study case, in SQ03, we asked our interviewees their opinion about whether some aspects negatively impacted team communication. Rare face-to-face interaction got 44% of answers saying agree or strongly agree about it, while getting 40% neither agree nor disagree, and 16% disagreed or strongly disagreed about it. These results converge to the same results found during the other phases of this study.

5.2.2.3 Less team interaction

When referencing less interaction between teams, adopting decoupled architecture may reduce the necessity of communication in a multi-site project, reducing inter-team communication [SM07]; this is more perceptible when teams work in different workflows (see quote EC-4). However, some authors affirm that it is necessary to have inter-team coordination to align modifications in the service interface or to keep integration (ILYAS; KHAN, 2017; ILYAS; KHAN, 2015; SIEVI-KORTE; BEECHAM; RICHARDSON, 2019).

Although decreasing the amount of communication may appear to improve the misunderstandings in multi-site teams, it also comes with drawbacks, like creating knowledge silos (see quote EA-1).

5.2.2.4 Agile ceremonies and management frameworks

Agile ceremonies can help to spread knowledge through distributed teams and establish a common understanding between them (see quote ED-4). Practices like pair programming or

daily standups are used in different companies to avoid communication problems by having frequent communication [SM04]. It is also necessary to have the business people and developers constantly communicating [SM11].

However, applying these practices is only possible in some situations because of team distribution and lack of timezone overlap (see quote EC-14). There are some strategies to overcome problems related to overlap working hours, like trying to increase the overlap working hours, reducing the meeting length, and choosing remote sites in the same or proximate time zones (SHRIVASTAVA; DATE, 2015).

During our case study, the interviewees also gave some perspective on the impact of agile practices on distributed team communication. In SQ1, we asked about multiple agile practices like automated testing, coding standards, collective code ownership, continuous integration, pair programming, refactoring, requirement workshop, scrum of scrum, incremental design, spring review/demo, test-driven development, and user stories. Through all these practices, we got good feedback, driving us to the same results obtained during the systematic mapping and the interviews.

5.2.2.5 Alignment meetings

Teams can take leverage alignment meetings to mitigate communication problems. There are different types of alignment meetings, for instance, check-in meetings (see quote EB-4), feedback meetings (see quotes EB-13 and EB-15), and requirement refinement (see quote EB-14). However, more than alignment meetings are needed in this process; having some direction in what practices must be followed by teams and which standards must be applied can drive complete confusion (see quote EC-13). Feedback is a crucial factor when talking about communication in distributed teams (see quotes EB7 and EB-5); it can also be used as a knowledge-sharing instrument (see quotes EB-16 and EB-19), helping build a reliable and open environment (see quotes EB-8 and EA-7). Companies sometimes adopt feedback meetings to discuss process and code improvements (MOE; STRAY; GOPLEN, 2020; CRUZ; JUNIOR; SARDINHA, 2021).

Having alignment meetings and continuous feedback practices is directly connected with the agile principles. As the agile manifesto presents in its values, responding to change is essential, which means that the means need to know about possible modifications or problems.

5.2.3 Culture

Our investigation identified cultural aspects from three perspectives: learning culture, transparency, and commitment. In the following sections, we will discuss these perspectives.

5.2.3.1 Learning culture

It is a complex aspect to control, but necessary to spread knowledge when adopting complicated architectures. The learning culture becomes even more critical when the team is composed of different seniority levels (see quote EA-9). Each person handles and absorbs knowledge differently, depending on the background (see quote EA-26); some individuals do not see the necessity to understand the basics behind frameworks, architecture (see quote EA-20), or tools used in the development process (see quote EA-25). It is necessary to comprehend the foundation to build a quality product (see quote EA-23). Furthermore, the learning culture is crucial to achieving an excellent shared understanding (see quote EA-19). However, it is challenging to create this culture in newcomers (see quote EA-6), regardless of the cases where the architecture complexity becomes challenging to maintain a good level of understanding (see quote EA-20).

The learning knowledge connects directly with knowledge-sharing practices and agile practices designed to achieve this purpose (ISLAM et al., 2012), like pair programming and sprint demos. Although knowledge-sharing is essential in distributed environments, distributed teams must accept architectural knowledge to make decisions abroad [SM02].

5.2.3.2 Transparency

Transparency is also related to knowledge-sharing and also open culture. To overcome challenges generated by distributed teams, they need to be completely transparent about their motivation for doing something (see quotes EB-8, EB-6, and EA-22). Having the information flow is essential to maintain every person at the same understanding level (see quotes EA-21 and EA-24).

An open and transparent environment enables building trust between team members (BETTA; BORONINA, 2018); the lack of trust is a known challenge in a distributed environment, increasing the time and budget and affecting product quality (HUMAYUN; JHANJHI, 2019).

In SQ03, we asked our interviewees whether a lack of trust hurt team communication, and 85% had a positive answer, corroborating our findings in the literature.

Understanding team skills are also necessary to keep transparency and build trust; not trusting each other may drive misunderstandings and increase the risk related to team interaction (see quotes EA-15, EA-16, and EA-5).

5.2.3.3 Commitment

Commitment is a cultural aspect that pops out during our investigation as a counterpoint to guidelines. It does not matter if the team has guidelines to follow. Commitment is necessary when practices are defined; just commitment will help the team members to follow these practices (see quote EA-14).

5.3 DESTILING OUR SUPPOSITIONS

This section will discuss the suppositions presented in section 5.1, using this study's results.

5.3.1 SP1: Decoupled software architectures are communication enablers and can help to mitigate communication challenges in DSD environments.

Software architecture is only sometimes simple to apply and even more complex in distributed software development environments. During this study, we have identified multiple architectural styles considered decoupled. Adopting decoupled architecture can work as a communication enabler.

Microservices are one of the multiple types that can be adopted, where the interfaces are split, making it easier to develop each service more independently. However, sometimes there are still dependencies between these microservices and require communication between teams to establish the development boundaries and better understand the business rules shared between these services.

Another architectural style that may be adopted is event-driven architecture, with a single point of communication and a more independent communication that improves failure handling. Adopting this type of architecture may help teams to have more independence. However, it also requires coordination between teams to establish a standard data contract to be followed

by everyone producing and consuming events from the environment.

No architecture is free of drawbacks and pitfalls when applied in a distributed software development environment. Adopting this type of architecture may be more beneficial to DSD environments, but it is necessary to be aware of the challenges related to adopting it.

5.3.2 SP2: An Architectural Design which enables agile practices and follows architectural-centric principles can help to coordinate DSD teams and mitigate communication challenges

Considering the benefits of applying agile principles and decoupled architecture, it is clear that adopting architectural-centric principles combined with agile practices benefits DSD teams.

Agile practices help to keep a good level of understanding between team members, and following it combined with an architecture that maintains a good structure can mitigate communication challenges. If the architecture is complex, it is necessary to establish a foundation knowledge from the start and build the project following it. It is essential to keep the same level of understanding during product development, including newcomers and old team members.

Although adopting architectural knowledge is not yet widely accepted by distributed software development teams, some references show that adopting it as a tool to improve the decision-making process may help avoid communication problems during the development process.

5.4 CAN ARCHITECTURAL DESIGN, WHICH ENABLES AGILE PRACTICES AND FOLLOWS ARCHITECTURAL-CENTRIC PRINCIPLES, HELP COORDINATE DSD TEAMS AND MITIGATE COMMUNICATION CHALLENGES?

Recapitulating our main research question, we explore how an architectural design that embraces agile practices and adheres to architectural-centric principles can effectively coordinate Distributed Software Development (DSD) teams while alleviating communication challenges. Our work reveals a series of compelling findings and insights that lend strong support to this notion:

Adopting architectural-centric principles is a linchpin for DSD teams, offering them a shared understanding and a common starting point. This alignment proves instrumental in

harmonizing efforts and reducing the potential for misunderstandings.

The importance of architectural transparency and an open culture cannot be overstated. This dynamic fosters trust among team members; certain architectural styles, such as decoupled architectures like microservices, emerge as communication facilitators. While challenges may arise, embracing decoupled architectures minimizes inter-team communication needs and bolsters team autonomy.

Furthermore, our findings underscore the significance of agile practices — daily standups, pair programming, and other agile ceremonies — in disseminating knowledge and enriching communication within distributed teams. These practices serve as conduits for ongoing collaboration and shared comprehension.

In addition, the cultivation of architectural knowledge-sharing and learning culture is essential. Equipping team members with a robust foundation in architectural concepts empowers them to make informed decisions, diminishing the likelihood of miscommunication.

Moreover, our research highlights the value of adhering to architectural guidelines and standards, encompassing coding practices and architectural patterns. This disciplined approach contributes to a more coherent and intelligible development process, particularly in the intricate landscape of a distributed environment.

The synergy between architectural-centric principles and agile practices collectively addresses multifaceted dimensions of communication, coordination, and shared understanding—imperative elements for the triumph of DSD. Acknowledging the potential for challenges, it is pivotal to emphasize that a steadfast architectural foundation, coupled with an agile mindset, can substantially mitigate these challenges, enhancing the overall effectiveness of DSD teams.

5.4.1 Lessons learned

The points listed below consolidate the lessons learned during the execution of this work, based on the results collected during the execution of this work and in the discussion.

- *Decoupled architecture mitigates communication challenges:* Decoupled architectures, such as microservices and event-driven architecture, can help reduce communication overhead and enable more independent development of components, making them suitable for distributed software development environments.

- *Consider pitfalls and drawbacks:* While adopting decoupled architectures can be beneficial, it is crucial to be aware of potential pitfalls and challenges, such as knowledge silos and coordination issues between teams.
- *Architecture and agile practices improve communication:* Combining architectural-centric principles with agile methods can help coordinate distributed teams and mitigate communication challenges. Having a foundation of architecture and following established practices can maintain a good understanding among team members.
- *Knowledge-sharing is essential:* Creating a learning culture and fostering knowledge-sharing practices are crucial for spreading architectural knowledge and achieving a shared understanding among team members.
- *Cultural aspects are critical:* Transparency and commitment are vital cultural aspects that can foster trust and open communication between distributed team members.
- *Continuous feedback builds trust:* Providing continuous feedback and open communication channels helps build trust among team members and improve communication.
- *Time zone proximity facilitates agile ceremonies:* Having teams working in the same or close time zones allows for more effortless execution of agile ceremonies and contributes to better trust-building within the team.

5.5 CLOSING REMARKS

Concluding this chapter, we emerge with a heightened understanding of our research landscape, forged through the crucible of systematic mapping, case study exploration, and reflective analysis. The amalgamation of these methodologies has yielded a multidimensional comprehension of our research domain, one that transcends quantitative data alone to embrace the qualitative essence of human experiences and perspectives. Through this synthesis, we've uncovered intricate connections, illuminated uncharted areas, and gained a comprehensive view of the factors that shape our field. Moreover, our journey has not only contributed to the existing academic discourse but has also revealed lessons learned in navigating the intricacies of research design, data collection, and interpretation. As we move forward, armed with these insights, we stand poised to delve deeper into unexplored realms, building upon this foundation to advance our understanding and contribute to scholarly conversation.

6 CONCLUSION

This thesis has provided valuable insights into the impact of software architecture on distributed software development teams. Through a systematic mapping and case study, several lessons have been learned that shed light on the challenges and opportunities that arise in such environments.

The findings highlight the significance of decoupled architectures, such as microservices and event-driven architecture, in alleviating communication challenges. These architectural approaches enable teams to work more independently on components, reducing communication overhead and promoting efficient development.

However, it is important to approach decoupled architectures with a nuanced understanding, as potential drawbacks like knowledge silos and coordination issues can emerge. Awareness of these pitfalls is essential for informed decision-making and effective implementation.

The study underscores the synergy between architectural-centric principles and agile practices in enhancing communication within distributed teams. By establishing a solid architectural foundation and adhering to agile methodologies, teams can better coordinate their efforts and navigate communication hurdles.

Furthermore, the research highlights the pivotal role of knowledge-sharing in fostering a shared understanding among team members. Cultivating a learning culture and promoting practices that facilitate the spread of architectural knowledge can significantly contribute to team cohesion and effectiveness.

Cultural aspects emerge as a critical factor in distributed teams, with transparency and commitment being key drivers of trust and open communication. Building trust among team members, in turn, is bolstered by continuous feedback and open communication channels, fostering an environment of collaboration and understanding.

Finally, the study emphasizes the value of time zone proximity in facilitating agile ceremonies and trust-building. When teams operate in the same or nearby time zones, the execution of agile practices becomes smoother, enabling better coordination and rapport among team members.

In essence, this research underscores that the success of distributed software development teams is deeply intertwined with the interplay of software architecture, agile practices, knowledge-sharing, cultural aspects, and geographical considerations. By embracing the lessons

learned from this study, organizations can navigate the complexities of distributed development and foster an environment of effective communication, collaboration, and innovation.

Some aspects that were considered possible threats to the validity of this work. The structure described in the points below was developed based on the aspects defined by Runeson and Höst (RUNESON; HÖST, 2009). They define four aspects: construct validity, internal validity, external validity, and reliability.

6.1 LIMITATIONS AND THREATS TO VALIDITY

This section presents the measures taken to address validity threats associated with the MSL in this study. Three types of validity threats, as described by Ampatzoglu et al. (AMPATZOGLOU et al., 2019), were identified and addressed through various analyses and actions. The following subsections detail the validity threats associated with the different activities of this study and the steps taken to mitigate them.

Internal validity: To identify the most considerable amount of papers and ensure good coverage of papers related to software architecture and DSD in hybrid environments, the search string uses multiple synonyms of software architecture, distributed teams, and hybrid and agile methodologies. Although we only searched four online digital libraries, they are supposed to cover most of the high-quality publications related to software engineering. In addition, even trying to avoid bias during the analysis, this study was not peer-reviewed during the extraction process. Another threat was during our interviews, the transcriptions and quotes presented in this work had to be translated into english when the interview was in a different language.

Construct Validity: To mitigate this threat, we conducted a peer review process. The first and second authors thoroughly examined each paper included in the study and discussed any discrepancies until a consensus was reached. Additionally, a third researcher was engaged to conduct an independent assessment of the mapping to ensure impartiality.

External Validity: Using a pre-defined search string in well-known bibliographic databases that cover references in agile development ensured that the findings have a certain level of generalizability, as most articles in the field are typically published in those databases.

6.2 FUTURE WORKS

In this work, we provide valuable insights into software architecture design in distributed software development, specifically in mitigating communication challenges. Our findings demonstrate the importance of software architecture in improving team performance, software quality, and communication. Furthermore, we generated two hypotheses through our discussion, which could lead to further investigations into the impact of software architecture design on real teams. While we found a limited number of studies in this area, our results suggest a promising avenue for future research.

In the list below, we describe some of the future works based on the results obtained from this work.

- *Quantitative Analysis of Communication Overhead Reduction:* Conduct a quantitative study to measure the actual reduction in communication overhead achieved by adopting decoupled architectures like microservices or event-driven architecture. Compare communication patterns and efficiency metrics before and after architecture changes.
- *Case Studies on Knowledge Silos Mitigation:* Perform in-depth case studies on organizations that have adopted decoupled architectures, focusing on how they address knowledge silos and coordination issues between teams. Analyze strategies, best practices, and real-world challenges.
- *Architectural Guidelines for Agile-Distributed Teams:* Develop a set of architectural guidelines specifically tailored to distributed agile teams. Investigate how architectural decisions can be aligned with agile principles to optimize communication and collaboration among team members.
- *Measuring Impact of Learning Culture:* Explore the correlation between strong learning culture and improved communication within distributed teams. Design surveys and gather data to measure how knowledge-sharing practices impact team dynamics and performance.
- *Cultural Influence on Architecture Adoption:* Investigate the role of cultural differences in the successful adoption of decoupled architectures. Examine how cultural aspects impact communication practices, and propose strategies to bridge cultural gaps within distributed teams.

- *Feedback Mechanisms for Trust Building:* Delve deeper into the types of feedback mechanisms that are most effective in building trust among distributed team members. Compare different feedback strategies and their impact on team communication and cohesion.
- *Long-Term Effects of Architectural Choices:* Investigate the long-term effects of architectural decisions on communication and team dynamics. Study projects with varying architectural approaches to understand how architectural choices impact collaboration over extended periods.
- *Tooling Support for Distributed Architecture Teams:* Evaluate existing and emerging tools that support communication and collaboration in distributed architecture teams. Investigate how these tools can enhance communication, architectural visualization, and decision-making processes.
- *Remote Pair Programming in Distributed Architectures:* Explore the feasibility and effectiveness of remote pair programming in distributed teams using decoupled architectures. Assess how this practice influences communication, knowledge transfer, and code quality.

REFERENCES

ALI, N.; BEECHAM, S.; MISTRIK, I. Architectural knowledge management in global software development: a review. In: IEEE. *2010 5th IEEE International Conference on Global Software Engineering*. [S.l.], 2010. p. 347–352.

ALZOUBI, Y. I.; GILL, A. Q. An agile enterprise architecture-driven model for geographically distributed agile development. In: *Transforming Healthcare Through Information Systems*. [S.l.]: Springer, 2016. p. 63–77.

ALZOUBI, Y. I.; GILL, A. Q. An empirical investigation of geographically distributed agile development: The agile enterprise architecture is a communication enabler. *IEEE Access*, IEEE, v. 8, p. 80269–80289, 2020.

ALZOUBI, Y. I.; GILL, A. Q.; MOULTON, B. A measurement model to analyze the effect of agile enterprise architecture on geographically distributed agile development. *Journal of Software Engineering Research and Development*, SpringerOpen, v. 6, n. 1, p. 1–24, 2018.

AMPATZOGLOU, A.; BIBI, S.; AVGERIOU, P.; VERBEEK, M.; CHATZIGEORGIOU, A. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and Software Technology*, v. 106, p. 201–230, 2019. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584918302106>.

ARANDA, G. N.; VIZCAÍNO, A.; PIATTINI, M. A framework to improve communication during the requirements elicitation process in gsd projects. *Requirements engineering*, Springer, v. 15, p. 397–417, 2010.

AVRITZER, A.; PAULISH, D.; CAI, Y.; SETHI, K. Coordination implications of software architecture in a global software development project. *Journal of Systems and Software*, Elsevier, v. 83, n. 10, p. 1881–1895, 2010.

BAHETI, P.; GEHRINGER, E.; STOTTS, D. Exploring the efficacy of distributed pair programming. In: SPRINGER. *Extreme Programming and Agile Methods—XP/Agile Universe 2002: Second XP Universe and First Agile Universe Conference Chicago, IL, USA, August 4–7, 2002 Proceedings 2*. [S.l.], 2002. p. 208–220.

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P. Microservices architecture enables devops. *London: Sharif University of Technology*, 2014.

BANO, M.; ZOWGHI, D.; SARKISSIAN, N. Empirical study of communication structures and barriers in geographically distributed teams. *IET software*, Wiley Online Library, v. 10, n. 5, p. 147–153, 2016.

BEAULIEU, N.; DASCALU, S. M.; HAND, E. Api-first design: A survey of the state of academia and industry. In: SPRINGER. *ITNG 2022 19th International Conference on Information Technology-New Generations*. [S.l.], 2022. p. 73–79.

BECK, K. *Extreme Programming: Embrace Change*. [S.l.]: Addison Wesley, 2001.

BECK, K. *Test-driven development: by example*. [S.l.]: Addison-Wesley Professional, 2003.

BECK, K.; BEEDLE, M.; BENNEKUM, A. V.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R. et al. *The agile manifesto*. 2001.

BELSHE, M.; PEON, R. *RFC 7540: hypertext transfer protocol version 2 (HTTP/2)*. [S.l.]: RFC Editor, 2015.

BETTA, J.; BORONINA, L. Transparency in project management–from traditional to agile. In: ATLANTIS PRESS. *Third International Conference on Economic and Business Management (FEBM 2018)*. [S.l.], 2018. p. 446–449.

BIEHL, M. *RESTful Api Design*. [S.l.]: API-University Press, 2016. v. 3.

BOSCH, J.; BOSCH-SIJTSEMA, P. Coordination between global agile teams: From process to architecture. In: *Agility Across Time and Space*. [S.l.: s.n.], 2010.

BOURQUE, P.; FAIRLEY, R. E. et al. *Guide to the Software Engineering Body of Knowledge (SWEBOK): Version 3.0*. [S.l.]: IEEE Computer Society Press, 2014.

BREIVOLD, H. P.; SUNDMARK, D.; WALLIN, P.; LARSSON, S. What does research say about agile and architecture? In: IEEE. *2010 Fifth International Conference on Software Engineering Advances*. [S.l.], 2010. p. 32–37.

CAMARA, R.; ALVES, A.; MONTE, I.; MARINHO, M. Agile global software development: A systematic literature review. In: *Proceedings of the 34th Brazilian Symposium on Software Engineering*. [S.l.: s.n.], 2020. p. 31–40.

CAMARA, R.; MONTE, I.; ALVES, A.; MARINHO, M. Hybrid practices in global software development: A systematic literature review. *International Journal of Software Engineering & Applications (IJSEA)*, v. 13, p. 1–17, 2022. ISSN 0975-9018.

CHAMAS, C. L.; CORDEIRO, D.; ELER, M. M. Comparing rest, soap, socket and grpc in computation offloading of mobile applications: An energy cost analysis. In: IEEE. *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*. [S.l.], 2017. p. 1–6.

CLARK, T.; BARN, B. S. Event driven architecture modelling and simulation. In: IEEE. *Proceedings of 2011 IEEE 6th International Symposium on Service Oriented System (SOSE)*. [S.l.], 2011. p. 43–54.

CLERC, V. Do architectural knowledge product measures make a difference in gsd? In: IEEE. *2009 Fourth IEEE International Conference on Global Software Engineering*. [S.l.], 2009. p. 382–387.

CLERC, V.; LAGO, P.; VLIET, H. V. Assessing a multi-site development organization for architectural compliance. In: IEEE. *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*. [S.l.], 2007. p. 10–10.

CLERC, V.; LAGO, P.; VLIET, H. V. Global software development: are architectural rules the answer? In: IEEE. *International Conference on Global Software Engineering (ICGSE 2007)*. [S.l.], 2007. p. 225–234.

CLERC, V.; LAGO, P.; VLIET, H. van. The usefulness of architectural knowledge management practices in gsd. In: IEEE. *2009 Fourth IEEE International Conference on Global Software Engineering*. [S.l.], 2009. p. 73–82.

CONWAY, M. E. How do committees invent. *Datamation*, v. 14, n. 4, p. 28–31, 1968.

CRNKOVIC, I. Component-based software engineering—new challenges in software development. *Software focus*, Wiley Online Library, v. 2, n. 4, p. 127–133, 2001.

CRUZ, E. F. C. d.; JUNIOR, F. E. F.; SARDINHA, E. D. An experience in the use of scrum and kanban for project development in a waterfall environment. In: *Proceedings of the XX Brazilian Symposium on Software Quality*. [S.l.: s.n.], 2021. p. 1–7.

DOGLIO, F. *Pro REST API Development with Node. js*. [S.l.]: Apress, 2015.

DUSTIN, E.; RASHKA, J.; PAUL, J. *Automated software testing: Introduction, management, and performance: Introduction, management, and performance*. [S.l.]: Addison-Wesley Professional, 1999.

EASTERBROOK, S.; SINGER, J.; STOREY, M.-A.; DAMIAN, D. Selecting empirical methods for software engineering research. *Guide to advanced empirical software engineering*, Springer, p. 285–311, 2008.

EBERT, C.; GALLARDO, G.; HERNANTES, J.; SERRANO, N. Devops. *Ieee Software*, IEEE, v. 33, n. 3, p. 94–100, 2016.

EDWARDS, C. Agile enterprise architecture, part 1. *USA: ProcessWave*, 2006.

ERL, T. *Service-oriented architecture*. [S.l.]: Citeseer, 1900.

EVANS, E. *Domain-driven design: tackling complexity in the heart of software*. [S.l.]: Addison-Wesley Professional, 2004.

EVANS, E. *Domain-Driven Design Reference: Definitions and Pattern Summaries*. [S.l.]: Dog Ear Publishing, 2014.

FARIA, H. R. D.; ADLER, G. Architecture-centric global software processes. In: IEEE. *2006 IEEE International Conference on Global Software Engineering (ICGSE'06)*. [S.l.], 2006. p. 241–242.

FAUZI, S. S. M.; BANNERMAN, P. L.; STAPLES, M. Software configuration management in global software development: A systematic map. In: IEEE. *2010 Asia Pacific Software Engineering Conference*. [S.l.], 2010. p. 404–413.

FOWLER, M.; FOEMMEL, M. *Continuous integration*. 2006.

FOWLER, M.; LEWIS, J. *Microservices*. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>.

FRANCESCO, P. D.; LAGO, P.; MALAVOLTA, I. Migrating towards microservice architectures: an industrial survey. In: IEEE. *2018 IEEE International Conference on Software Architecture (ICSA)*. [S.l.], 2018. p. 29–2909.

GLASER, B. G.; STRAUSS, A. L.; STRUTZEL, E. The discovery of grounded theory; strategies for qualitative research. *Nursing research*, LWW, v. 17, n. 4, p. 364, 1968.

GUDGIN, M.; HADLEY, M.; MENDELSOHN, N.; MOREAU, J.-J.; NIELSEN, H. F.; KARMARKAR, A.; LAFON, Y. *SOAP Version 1.2*. [S.l.]: w3c, 2003.

HERBSLEB, J. D. Global software engineering: The future of socio-technical coordination. In: IEEE. *Future of Software Engineering (FOSE'07)*. [S.l.], 2007. p. 188–198.

HERBSLEB, J. D.; GRINTER, R. E. Architectures, coordination, and distance: Conway's law and beyond. *IEEE software*, IEEE, v. 16, n. 5, p. 63–70, 1999.

HERBSLEB, J. D.; MOITRA, D. *Global software development*. [S.l.]: World Scientific, 2015. v. 4.

HIGHSMITH, J. A.; HIGHSMITH, J. *Agile software development ecosystems*. [S.l.]: Addison-Wesley Professional, 2002.

HOLMSTRÖM, H.; FITZGERALD, B.; ÅGERFALK, P. J.; CONCHÚIR, E. Ó. et al. Agile practices reduce distance in global software development. *Information systems management*, Taylor & Francis, v. 23, n. 3, p. 7–18, 2006.

HUMAYUN, M.; JHANJHI, N. Exploring the relationship between gsd, knowledge management, trust and collaboration. *Journal of Engineering Science and Technology*, v. 14, n. 2, p. 820–843, 2019.

HUMBLE, J.; FARLEY, D. *Continuous delivery: reliable software releases through build, test, and deployment automation*. [S.l.]: Pearson Education, 2010.

IBM. *Innovation in the API economy: Building winning experiences and new capabilities to compete*. 2016.

ILYAS, M.; KHAN, S. U. Software integration in global software development: Success factors for gsd vendors. In: IEEE. *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. [S.l.], 2015. p. 1–6.

ILYAS, M.; KHAN, S. U. An empirical investigation of the software integration success factors in gsd environment. In: IEEE. *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*. [S.l.], 2017. p. 255–262.

ISLAM, T.; ANWAR, F.; KHAN, S. U. R.; RASLI, A.; AHMAD, U.; AHMED, I. Investigating the mediating role of organizational citizenship behavior between organizational learning culture and knowledge sharing. *World Applied Sciences Journal*, v. 19, n. 6, p. 795–799, 2012.

ISO/IEC/IEEE International Standard - Systems and software engineering – Software life cycle processes. *ISO/IEC/IEEE 12207:2017(E) First edition 2017-11*, p. 1–157, 2017.

JACOBSON, D.; BRAIL, G.; WOODS, D. *APIs: A strategy guide*. [S.l.]: O'Reilly Media, Inc., 2012.

JAIN, R.; SUMAN, U. Effectiveness of agile practices in global software development. *International Journal of Grid and Distributed Computing*, v. 9, n. 10, p. 231–248, 2016.

JAN, S. R.; DAD, F.; AMIN, N.; HAMEED, A.; SHAH, S. S. A. Issues in global software development (communication, coordination and trust) a critical review. *training*, v. 6, n. 7, p. 8, 2016.

JUNIOR, I. d. F.; MARCZAK, S.; SANTOS, R.; RODRIGUES, C.; MOURA, H. C2m: a maturity model for the evaluation of communication in distributed software development. *Empirical Software Engineering*, Springer, v. 27, n. 7, p. 188, 2022.

KHAN, A. A.; BASRI, S.; DOMINC, P. A proposed framework for communication risks during rcm in gsd. *Procedia-Social and Behavioral Sciences*, Elsevier, v. 129, p. 496–503, 2014.

KIRCHER, M.; JAIN, P.; CORSARO, A.; LEVINE, D. Distributed extreme programming. *Extreme Programming and Flexible Processes in Software Engineering, Italy*, p. 66–71, 2001.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. 2007.

KORNSTÄDT, A.; SAUER, J. Mastering dual-shore development–the tools and materials approach adapted to agile offshoring. In: SPRINGER. *International Conference on Software Engineering Approaches for Offshore and Outsourced Development*. [S.l.], 2007. p. 83–95.

KORNSTADT, A.; SAUER, J. Tackling offshore communication challenges with agile architecture-centric development. In: IEEE. *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*. [S.l.], 2007. p. 28–28.

KULKARNI, G. Cloud computing-software as service. *International Journal of Cloud Computing And Services Science*, IAES Institute of Advanced Engineering and Science, v. 1, n. 1, p. 11, 2012.

LEFFINGWELL, D. *SAFe 4.5 reference guide: scaled agile framework for lean enterprises*. [S.l.]: Addison-Wesley Professional, 2018.

LENARDUZZI, V.; SIEVI-KORTE, O. On the negative impact of team independence in microservices software development. In: *Proceedings of the 19th International Conference on Agile Software Development: Companion*. [S.l.: s.n.], 2018. p. 1–4.

LI, H. Restful web service frameworks in java. In: IEEE. *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*. [S.l.], 2011. p. 1–4.

MA, X.; LIU, Y. An empirical study of maven archetype. In: *SEKE*. [S.l.: s.n.], 2020. p. 153–157.

MALAVOLTA, I.; CAPILLA, R. Current research topics and trends in the software architecture community: Icsa 2017 workshops summary. In: IEEE. *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. [S.l.], 2017. p. 1–4.

MARÉCHAUX, J.-L. Combining service-oriented architecture and event-driven architecture using an enterprise service bus. *IBM developer works*, v. 12691275, 2006.

MARINHO, M.; CAMARA, R.; SAMPAIO, S. Toward unveiling how safe framework supports agile in global software development. *IEEE Access*, IEEE, v. 9, p. 109671–109692, 2021.

MARINHO, M.; NOLL, J.; BEECHAM, S. Uncertainty management for global software development teams. In: IEEE. *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*. [S.l.], 2018. p. 238–246.

MARINHO, M.; NOLL, J.; RICHARDSON, I.; BEECHAM, S. Plan-driven approaches are alive and kicking in agile global software development. In: *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Porto de Galinhas, Brazil: IEEE, 2019. p. 1–11.

MARTIN, R. C. *Clean architecture*. [S.l.]: Prentice Hall, 2017.

MAY, I. *Systems and software engineering–architecture description*. [S.l.], 2011.

MICROSYSTEMS, S. *RPC: Remote Procedure Call Protocol Specification Version 2*. 2009. Disponível em: <https://www.rfc-editor.org/rfc/rfc5531>.

MISHRA, A.; MISHRA, D. Software architecture in distributed software development: A review. In: SPRINGER. *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. [S.l.], 2013. p. 284–291.

MOE, N. B.; STRAY, V.; GOPLEN, M. R. Studying onboarding in distributed software teams: a case study and guidelines. In: *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*. [S.l.: s.n.], 2020. p. 150–159.

NEWCOMER, E.; LOMOW, G. *Understanding SOA with Web services*. [S.l.]: Addison-Wesley, 2005.

NEWMAN, S. *Building microservices*. [S.l.]: "O'Reilly Media, Inc.", 2021.

NOLL, J.; BEECHAM, S.; RICHARDSON, I. Global software development and collaboration: barriers and solutions. *ACM inroads*, ACM New York, NY, USA, v. 1, n. 3, p. 66–78, 2011.

NORD, R. L.; TOMAYKO, J. E. Software architecture-centric methods and agile development. *IEEE software*, IEEE, v. 23, n. 2, p. 47–53, 2006.

NOSEK, J. T. The case for collaborative programming. *Communications of the ACM*, ACM New York, NY, USA, v. 41, n. 3, p. 105–108, 1998.

OVASKA, P.; ROSSI, M.; MARTTIIN, P. Architecture as a coordination tool in multi-site software development. *Software Process: Improvement and Practice*, Wiley Online Library, v. 8, n. 4, p. 233–247, 2003.

PAGE-JONES, M. *The Practical Guide to Structured Systems Design: 2nd Edition*. USA: Yourdon Press, 1988. ISBN 0136907695.

PAUTASSO, C. Restful web service composition with bpel for rest. *Data & Knowledge Engineering*, Elsevier, v. 68, n. 9, p. 851–866, 2009.

PERREY, R.; LYCETT, M. Service-oriented architecture. In: *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.* [S.l.: s.n.], 2003. p. 116–119.

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, ACM, v. 17, n. 4, p. 40–52, 1992.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*. [S.l.: s.n.], 2008. p. 1–10.

PETROV, V.; AZALETSKIY, P. Socio-informational system contradictions and evolution laws. *TRIZ in Evolution*, . . . , n. 1, p. 265–275, 2023.

QIAN, L.; LUO, Z.; DU, Y.; GUO, L. Cloud computing: An overview. In: SPRINGER. *IEEE International Conference on Cloud Computing*. [S.l.], 2009. p. 626–631.

QURASHI, S. A.; QURESHI, M. R. J. Scrum of scrums solution for large size teams using scrum methodology. arXiv, 2014. Disponível em: <https://arxiv.org/abs/1408.6142>.

RÄTY, P.; BEHM, B.; DIKERT, K.-K.; PAASIVAARA, M.; LASSENIUS, C.; DAMIAN, D. Communication practices in a distributed scrum project. In: *The 4th International Conference on Collaborative Innovation Networks COINs13, Santiago de Chile, August 11-13 2013*. [S.l.: s.n.], 2013.

RICHARDSON, L.; AMUNDSEN, M.; RUBY, S. *RESTful Web APIs: Services for a Changing World*. [S.l.]: " O'Reilly Media, Inc.", 2013.

RIGBY, M. *Component-Based Software Engineering: Software Architecture*. [S.l.]: CreateSpace Independent Publishing Platform, 2016.

ROBSON, C. Real world research blackwell. *2º edição*, 2002.

RUNESON, P.; HÖST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, Springer, v. 14, n. 2, p. 131, 2009.

SALAMEH, A.; BASS, J. M. Heterogeneous tailoring approach using the spotify model. In: *Proceedings of the Evaluation and Assessment in Software Engineering*. [S.l.: s.n.], 2020. p. 293–298.

SAUER, J. Architecture-centric development in globally distributed projects. In: *Agility Across Time and Space: Implementing Agile Methods in Global Software Projects*. [S.l.]: Springer, 2010. p. 321–329.

SCHREYER, A. C. *Architectural Design with SketchUp: Component-based Modeling, Plugins, Rendering, and Scripting*. [S.l.]: John Wiley & Sons, 2012.

SCHWABER, K.; SUTHERLAND, J. The scrum guide. *Scrum Alliance*, v. 21, n. 1, p. 1–38, 2011.

Scientific Software Development GmbH. *ATLAS.ti*. Disponível em: <https://atlasti.com/>.

SHRIVASTAVA, S. V.; DATE, H. A framework for risk management in globally distributed agile software development (agile gsd). *differences*, v. 4, n. 3, p. 97–111, 2015.

SIEVI-KORTE, O.; BEECHAM, S.; RICHARDSON, I. Challenges and recommended practices for software architecting in global software development. *Information and software technology*, Elsevier, v. 106, p. 234–253, 2019.

SIEVI-KORTE, O.; RICHARDSON, I.; BEECHAM, S. Software architecture design in global software development: An empirical study. *Journal of Systems and Software*, Elsevier, v. 158, p. 110400, 2019.

TAYLOR, R. N.; MEDVIDOVIC, N.; ANDERSON, K. M.; WHITEHEAD, E. J.; ROBBINS, J. E.; NIES, K. A.; OREIZY, P.; DUBROW, D. L. A component-and message-based architectural style for gui software. *IEEE Transactions on Software Engineering*, IEEE, v. 22, n. 6, p. 390–406, 1996.

TEKINERDOGAN, B.; CETIN, S.; BABAR, M. A.; LAGO, P.; MÄKIÖ, J. Architecting in global software engineering. *ACM SIGSOFT Software Engineering Notes*, ACM New York, NY, USA, v. 37, n. 1, p. 1–7, 2012.

URREGO, J.; MUÑOZ, R.; MERCADO, M.; CORREAL, D. Archinotes: A global agile architecture design approach. In: SPRINGER. *International Conference on Agile Software Development*. [S.l.], 2014. p. 302–311.

VALLON, R.; ESTÁCIO, B. J. da S.; PRIKLADNICKI, R.; GRECHENIG, T. Systematic literature review on agile practices in global software development. *Information and Software Technology*, Elsevier, v. 96, p. 161–180, 2018.

VANZIN, M.-A.; RIBEIRO, M. B.; PRIKLADNICKI, R.; CECCATO, I.; ANTUNES, D. Global software processes definition in a distributed environment. In: IEEE. *29th Annual IEEE/NASA Software Engineering Workshop*. [S.l.], 2005. p. 57–65.

VARANASI, B.; BELIDA, S.; VARANASI, B.; BELIDA, S. Maven archetypes. *Introducing Maven*, Springer, p. 47–62, 2014.

VLIET, H. V. Software architecture knowledge management. In: IEEE. *19th australian conference on software engineering (aswec 2008)*. [S.l.], 2008. p. 24–31.

WANG, X.; ZHAO, H.; ZHU, J. Grpc: A communication cooperation mechanism in distributed systems. *ACM SIGOPS Operating Systems Review*, ACM New York, NY, USA, v. 27, n. 3, p. 75–86, 1993.

WILLIAMS, L. The collaborative software process phd dissertation. *Department of Computer Science*, 2000.

WOLFF, E. *Microservices: flexible software architecture*. [S.l.]: Addison-Wesley Professional, 2016.

YANG, C.; LIANG, P.; AVGERIOU, P. A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, Elsevier, v. 111, p. 157–184, 2016.

YILDIZ, B. M.; TEKINERDOGAN, B.; CETIN, S. *A tool framework for deriving the application architecture for global software development projects*. 2012. 94–103 p.

APPENDIX A – SELECTED PAPERS

Id	Title	Ref
1	*Archinotes: A Global Agile Architecture Design Approach*	(URREGO et al., 2014)
2	*Do Architectural Knowledge Product Measures Make a Difference in GSD?*	(CLERC, 2009)
3	*Global Software Development: Are Architectural Rules the Answer?*	(CLERC; LAGO; VLIET, 2007b)
4	*Mastering Dual-Shore Development – The Tools and Materials Approach Adapted to Agile Offshoring*	(KORNSTÄDT; SAUER, 2007)
5	*Tackling Offshore Communication Challenges with Agile Architecture-Centric Development*	(KORNSTADT; SAUER, 2007)
6	*On the negative impact of team independence in microservices software development*	(LENARDUZZI; SIEVI-KORTE, 2018)
7	*Software Architecture in Distributed Software Development: A Review*	(MISHRA; MISHRA, 2013)
8	*Architecting in Global Software Engineering*	(TEKINERDOGAN et al., 2012)
9	*Architecture-centric Development in Globally Distributed Projects*	(SAUER, 2010)
10	*Software architecture design in global software development: An empirical study*	(SIEVI-KORTE; RICHARDSON; BEECHAM, 2019)
11	*An Agile Enterprise Architecture-Driven Model for Geographically Distributed Agile Development*	(ALZOUBI; GILL, 2016)
12	*A measurement model to analyze the effect of agile enterprise architecture on geographically distributed agile development*	(ALZOUBI; GILL; MOULTON, 2018)
13	*An Empirical Investigation of Geographically Distributed Agile Development: The Agile Enterprise Architecture Is a Communication Enabler*	(ALZOUBI; GILL, 2020)

APPENDIX B – INTERVIEW QUESTIONS

Code	Question
IQ01	Which architectural practices/process do you think that could improve communication between team members spread across multiple locations/countries? Why do you think they could help ?
IQ02	Do you think that architectural practices/rules can help to avoid misunderstandings ? why ?
IQ03	Would you link to highlight any development or architectural practice that you think can help to mitigate miscommunication ?

APPENDIX C – SURVEY QUESTIONS

C.1 PROCESSES AND PRACTICES

Question Code:	SQ01
Question:	Which of the following practices do you use?
Alternative Type:	**Likert Scale** 1 = We never use it 2 = We rarely use it 3 = We sometimes use it 4 = We often use it 5 = We always use the practice
Alternatives:	SQ01A01 = Automated Testing SQ01A02 = Coding Standards SQ01A03 = Collective Code Ownership SQ01A04 = Continuous Integration SQ01A05 = Pair Programming SQ01A06 = Refactoring SQ01A07 = Requirements Workshop SQ01A08 = Scrum of Scrum SQ01A09 = Simple/Incremental Design SQ01A10 = Sprint review/demo SQ01A11 = Test Driven Development SQ01A12 = User stories

Question Code:	SQ02
Question:	Which of the following architectural rules/designs/frameworks do you use?
Alternative Type:	**Likert Scale** 1 = We never use it 2 = We rarely use it 3 = We sometimes use it 4 = We often use it 5 = We always use the practice
Alternatives:	SQ02A01 = Application Programming Interface SQ02A02 = Domain-Driven Design SQ02A03 = Event-Driven Architecture SQ02A04 = Microservices SQ02A05 = Model-Driven Design SQ02A06 = REST/RESTful SQ02A07 = Service-Oriented Architecture SQ02A08 = Component-based architecture

Question Code:	SQ03
Question:	The following aspects have a bad impact on team communication.
Alternative Type:	**Likert Scale** 1 = Strongly agree 2 = Agree 3 = Neither agree nor disagree 4 = Disagree 5 = Strongly disagree
Alternatives:	SQ03A01 = Lack of skills SQ03A02 = Lack of trust SQ03A03 = Language limitation SQ03A04 = Not being aware of cultural differences SQ03A05 = Poor communication SQ03A06 = Poor documentation SQ03A07 = Rare face-to-face interaction

C.2 IMPACT OF SOFTWARE ARCHITECTURE ON DISTRIBUTED TEAMS

Question Code:	SQ04
Question:	Software architecture can positively impact software development teams working in multiple locations.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ05
Question:	Adopting micro services or/and micro frontends can help improve communication between team members in multiple locations.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ06
Question:	Event-driven architecture can help improve communication between team members in multiple locations.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ07
Question:	Component-based architecture can help improve communication between team members in various places.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ08
Question:	Domain-Driven Design can help improve the communication between team members on numerous sites.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ09
Question:	A decoupled component is a solution to mitigate communication challenges.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ10
Question:	Microservices generate decoupled components.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ11
Question:	Having decoupled components architecture reduces the necessity for inter-team communication.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ12
Question:	Architectural knowledge improves performance on distributed teams.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ13
Question:	Micro services can provide a better component overview and enable less coupling between applications.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ14
Question:	The knowledge transfer practices (like pair programming, components documentation, shadowing, etc.) help to reduce communication problems on distributed teams.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ15
Question:	Using APIs (Application Programming Interfaces) generates less interdependency between components and enables micro services.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ16
Question:	Centralized architectural modifications help to avoid communication challenges.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

Question Code:	SQ17
Question:	Architectural knowledge improves distributed teams' performance.
Alternative Type:	**Likert Scale** 1 = Totally disagree 2 = Partially disagree 3 = I don't know 4 = Partially agree 5 = Totally agree
Alternatives:	1 to 5

C.3 DEMOGRAPHIC QUESTIONS

Question Code:	SQ18
Question:	Which country are you located in?
Alternative Type:	**Multiple Choice**
Alternatives:	SQ18A01 = Spain SQ18A02 = Germany SQ18A03 = India SQ18A04 = Portugal SQ18A05 = Others

Question Code:	SQ19
Question:	What is your major role in your current project?
Alternative Type:	**Multiple Choice**
Alternatives:	SQ19A01 = Software Developer SQ19A02 = Team Lead SQ19A03 = Quality Analyst SQ19A04 = Software Architect SQ19A05 = Others

Question Code:	SQ20
Question:	What's your current education level? (consider the most recent completed)
Alternative Type:	**Multiple Choice**
Alternatives:	SQ20A01 = High School degree SQ20A02 = Technical degree SQ20A03 = Bachelor's degree SQ20A04 = Master's degree SQ20A05 = Doctorate degree SQ20A06 = Others

Question Code:	SQ21
Question:	How many years of experience do you have in software and systems development?
Alternative Type:	**Multiple Choice**
Alternatives:	SQ21A01 = < 1 year SQ21A02 = 1 - 2 years SQ21A03 = 3 - 5 years SQ21A04 = 6 - 10 years SQ21A05 = > 10 years

Question Code:	SQ22
Question:	Are there people on your team that came from a country that is not the same as you?
Alternative Type:	**Multiple Choice**
Alternatives:	SQ22A01 = Yes, 1 person SQ22A02 = Yes, 2-3 people SQ22A03 = Yes, more than 3 people SQ22A04 = No

Question Code:	SQ23
Question:	How many people your current team has?
Alternative Type:	**Multiple Choice**
Alternatives:	SQ23A01 = < 5 people SQ23A02 = 6 - 10 people SQ23A03 = 11 - 15 people SQ23A04 = 16 - 20 people SQ23A05 = > 20 people

Question Code:	SQ23
Question:	How many people your current team has?
Alternative Type:	**Multiple Choice**
Alternatives:	SQ23A01 = < 5 people SQ23A02 = 6 - 10 people SQ23A03 = 11 - 15 people SQ23A04 = 16 - 20 people SQ23A05 = > 20 people

Question Code:	SQ24
Question:	Please write below if you want to add any other information to this research.
Alternative Type:	**Open answer**
Alternatives:	None

APPENDIX D – EXECUTIVE SUMMARY

Executive Summary

Summary development by Thiago Gomes based on "The software architecture challenges in Agile Distributed Software Development environments" dissertation presented as partial requirement to obtain the master's degree in applied informatics from the Federal Rural University of Pernambuco (UFRPE).

Study Overview

The original study had the objective to study the challenges related to software architecture design in a software development environment where the team members are spread across multiple locations which drives to challenges related to communication and coordination.

Introduction

The study cited before was conducted in two phases, first we analyze the state of art related to the methods and challenges connected between managing distributed software development teams and how software architecture impacts the coordination and communication. The second phase we analyze the market perspective about the communication between team members spread in multiple location and relate these perspectives with practical technics that could help to mitigate communication challenges.

Outcome

As outcome from this analysis, we extract a set of lessons learned during the execution process which can be employed to improve the dynamics in distributed software development teams.

The main aspects surrounded by the lessons learned are software architecture design, challenges awareness and continuous communication. The challenges are described below.

1. Decoupled architecture mitigates communication challenges: Decoupled architectures, such as microservices and event-driven architecture, can help reduce communication overhead and enable more independent development of components, making them suitable for distributed software development environments.

2. Consider pitfalls and drawbacks: While adopting decoupled architectures can be beneficial, it is crucial to be aware of potential pitfalls and challenges, such as knowledge silos and coordination issues between teams.

3. Architecture and agile practices improve communication: Combining architectural-centric principles with agile methods can help coordinate

distributed teams and mitigate communication challenges. Having a foundation of architecture and following established practices can maintain a good understanding among team members.

4. Knowledge-sharing is essential: Creating a learning culture and fostering knowledge sharing practices are crucial for spreading architectural knowledge and achieving a shared understanding among team members.

5. Cultural aspects are critical: Transparency and commitment are vital cultural aspects that can foster trust and open communication between distributed team members.

6. Continuous feedback builds trust: Providing continuous feedback and open communication channels helps build trust among team members and improve communication.

7. Time zone proximity facilitates agile ceremonies: Having teams working in the same or close time zones allows for more effortless execution of agile ceremonies and contributes to better trust-building within the team.

Conclusions and Final Considerations

Each case scenario and each team have its own challenges, so it's important to evaluate how each one of this aspect can be applied in every situation. In most cases, challenge awareness can help to mitigate future problems, being aware is the first step to understand and avoid problems related to miscommunication or misunderstanding.

All the lessons learned presented before can be used as reference to build your own strategy to mitigate problems in distributed software development teams.