AGILE SOFTWARE ARCHITECTURE INGLOBAL SOFTWARE DEVELOPMENT ENVIRONMENT:SYSTEMATIC LITERATURE MAPPING

Thiago Gomes[1], and Marcelo Marinho[1]

[1]Master degree Program in Applied Informatics, Federal Rural University of Pernambuco, Recife, Pernambuco, BR

ABSTRACT

In recent years, software development companies started to adopt Global Software Development (GSD) to explore the benefits of this approach, mainly cost reduction. However, the GSD environment also brings more complexity and challenges. Some challenges are related to communication aspects like cultural differences, time zone, and language. This paper is the first step in an extensive study to understand if the software architecture can ease communication in GSD environments. We conducted a Systematic Literature Mapping (SLM) to catalog relevant studies about software architecture and GSD teams and identify potential practices for use in the software industry. This paper's findings contribute to the GSD body of knowledge by exploring the impact of software architecture strategy on the GSD environment. It presents hypotheses regarding the relationship between software architecture and GSD challenges, which will guide future research.

KEYWORDS

Global Software Development, software architecture, software architecture design

1. INTRODUCTION

In recent years, software development companies started to adopt Global Global Software Development (GSD) as an increasingly popular approach for reducing software project costs by leveraging talent worldwide. However, this approach introduces significant complexity when managing individuals from diverse cultural backgrounds, working across different time zones, and communicating in multiple languages [1]. These challenges can create issues impacting stakeholder collaboration throughout the development lifecycle. Poor communication and misunderstandings can lead to project delays, budget overruns, and diminished software quality [2].

In recent years, we could see many studies about GSD related to software project management, development process, and organizational factors [3, 4]. However, small contributions exist when we look into studies on software architecture related to GSD [3]. So, our paper focuses on identifyinga relationship between communication challenges and software architecture design.

This paper identifies ways to mitigate communication challenges inside the GSD team who specifically adopts Agile practices. A decoupled architecture can help these teams since communication between teams using these types of architectures is better structured. On the other hand, it is necessary to understand how to apply these architectures.

DOI: 10.5121/ijsea.2023.14301

In order to do that, we conducted a Systematic Literature Mapping (SLM) [5] that focused on finding research approaches from 2003 to 2020 that highlighted the agile software architecture in GSD environments.

This paper contributes to the GSD body of knowledge by presenting a set of findings: (i) the relation between software architecture strategy and its impact on the GSD environment; (ii) some hypotheses related to the impact of software architecture over the GSD challenges, which are going drive future studies and try to validate it; (iii) mindmaps relating different aspect existing on the literature with software architecture aspects.

The remainder of this paper is structured as follows: in Section 2, we introduce the background to the problem; Section 3 brings the explanation of our method, Section 4 presents the findings of this research, and Section 5 presents a discussion about the results and what we can do with these results. Finally, in Section 6, we state the threats to the validity of this study, and Section 7 presents the conclusions of this research and the opportunities for future works.

2. BACKGROUND

2.1. Agile Software Development

The "Agile Movement" first came to light with the Agile Manifesto, published by software consultants and practitioners in 2001 [6]. The focus was to bring more importance to the human aspects over the processes during the software development cycle [6]. The agile methods have much in common, with the same scaffold, but differ by adopted practices. Extreme Programming (XP) [6], Scrum [7], and Lean Development [8] are examples of agile methods and frameworks. Some methods and frameworks also focus on agile at a large scale, like SAFe® [9], Scrum of Scrums (SoS) [10], and The Spotify Model [11].

2.2. Global software development

Global Software Development (GSD) is a term used to describe the following situations: organizations that move all or part of their software development team to low-cost regions or regions where exists a better availability of the necessary skills; organizations that spread on multiple countries their software development teams [2]. An organization may develop the software worldwide for use, sale, or incorporation into a company's product (in-sourcing). A company may outsource the software development process to a supplier in the same country or a different country, which, as follow, develop the software for the client (outsourcing) [2].

2.3. Software Architecture Design

The software architecture modeling includes components and interfaces [12], which interconnect multiple structures [13]. The software architecture enables project coordination [14], for colocated teams and as a mechanism to allocate tasks and coordinate distributed teams [15].

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 enablesa better understanding of the business domain regarding cultural differences [16].

Software architecture can assume multiple shapes, such as a layered structure or a type of structured pipeline. It can interact as the message-based architecture [17], or service-based following

the service-oriented architecture (SOA) [18], the RESTful approach [19], or even one of the recent tendencies, denominated microservices [20, 21]. The microservice approach evolved based on the increasing demand for cloud computing [22, 23] and following the *aaS (as a Service) structure.

3. METHOD

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

We conducted a SLM, following the guidelines proposed by Petersen et al. [5]. First, we started defining the research questions that would 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 3.4

3.1. Research questions

We sought to answer the following research questions: [RQ1] How software architecture design impacts the GSD environment? and [RQ2] Is there any architectural design that can positively impact the GSD environment?

3.2. Inclusion and exclusion criteria

The following criteria guided the selection of papers that helped us address the research questions.

We *included*: (i) Studies that approach using software architecture inside of the GSD environment; and (ii) Papers which the keywords appear on Abstract and/or Author keywords.

We *excluded*: (i) Papers not related to Global Software Development and Software Architecture Design at the same time; (ii) Studies related to teaching global software development; (iii) Studies which focus is not software architecture design; and (iv) Studies not written in English.

3.3. 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 the first paper found related to decoupled-resilient software architectures, written by Perrey and Lycett [24].

To identify a set of relevant papers for our study, we conducted searches using a targeted set of keywords. Our search strategy began with examining top-ranked hits using simple keywords, which may have been overlooked by the final complex search string. We started with general keywords such as Software Architecture, Microservices, GSD, 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 archi-

tecture" 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))

3.4. 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 3.3, 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. Ultimately, we selected 13 studies as our final set of papers. The entire selection process is illustrated in Figure 1.

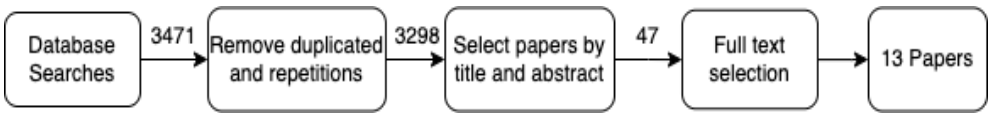

Figure 1. Selection process

3.5. Data Extraction and Analysis

Once we had our set of selected papers, we used ATLAS.ti [25] to analyze them. Each study was meticulously examined, and we extracted quotes and 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 section 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.

4. RESULTS

In this section, we will present our results, which have been grouped into three main categories: communication, software architecture, and solutions to mitigate challenges in GSD, and their interrelationships. We will refer to the selected papers (Table 1) using the SMXX format, where XX represents the paper ID with two digits.

Table 1. Selected papers (SM-ID)

Id	Title	Ref
1	*Archinotes: A Global Agile Architecture Design Approach*	[26]
2	*Do Architectural Knowledge Product Measures Make a Difference in GSD?*	[27]
3	*Global Software Development: Are Architectural Rules the Answer?*	[28]
4	*Mastering Dual-Shore Development The Tools and Materials Approach Adapted to Agile Offshoring*	[29]
5	*Tackling Offshore Communication Challenges with Agile Architecture-Centric Development*	[30]
6	*On the negative impact of team independence in microservices software development*	[31]
7	*Software Architecture in Distributed Software Development: A Review*	[32]

8	*Architecting in Global Software Engineering*	[33]
9	*Architecture-centric Development in Globally Distributed Projects*	[34]
10	*Software architecture design in global software development: An empirical study*	[35]
11	*An Agile Enterprise Architecture-Driven Model for Geographically Distributed Agile Development*	[36]
12	*A measurement model to analyze the effect of agile enterprise architecture on geographically distributed agile development*	[37]
13	*An Empirical Investigation of Geographically Distributed Agile Development: The Agile Enterprise Architecture Is a Communication Enabler*	[38]

4.1. Microservices and Communication

On the first mindmap (see Figure 2), 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 [39].

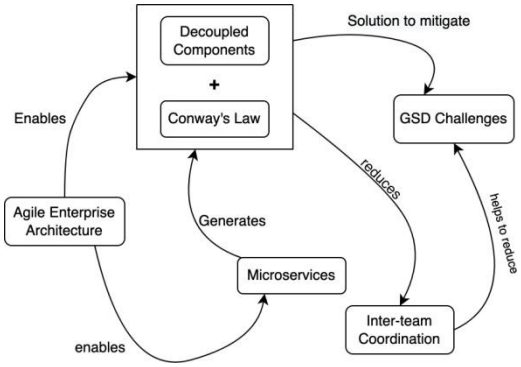

Figure 2. Communication and Decoupled-components mindmap

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

Alzoubi and Gill [SM13] highlight that using AEA as a typical model between the GSD 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: (i) *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; (ii) *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; and *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) [40, 41] may enable Microservices architecture [42], which can help the teams achieve the benefits generated by using decoupled components when Conway's law is applied. [SM13]

4.2. Architectural-centric development and performance on distributed teams

On the second mindmap (see Figure 3), 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.

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 GSD 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

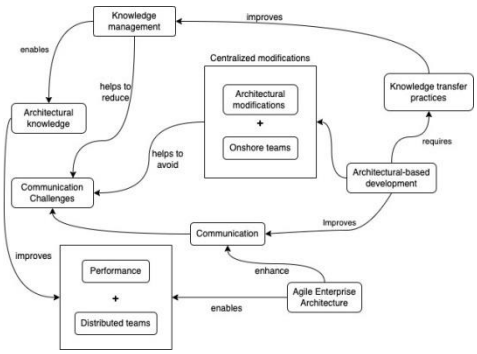

Figure 3. Agile principles and Communication

a study of cases where they analyze two organizations and discuss possible solutions to mitigate some GSD challenges. They found that organizations use a wiki, highly communicative meetings, and subsystem websites to reduce difficulty exchanging information. Furthermore, both organi-

zations 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: (i) *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; (ii) 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; and (iii) *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 in additional areas like 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.

4.3. Agile principles and GSD communication

On the third mindmap (see Figure 4), 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 practicesis 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.

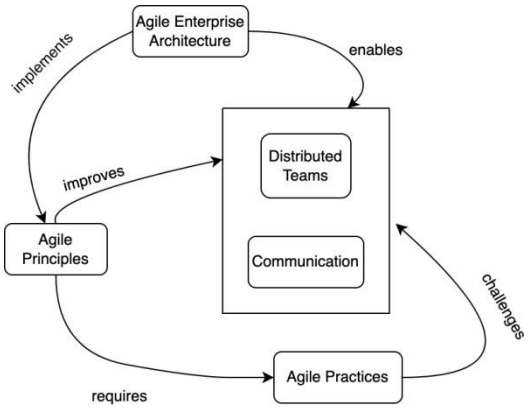

Figure 4. Architectural-centric development and communication

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 view 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.

4.4. Losely coupled components and communication challenges on GSD

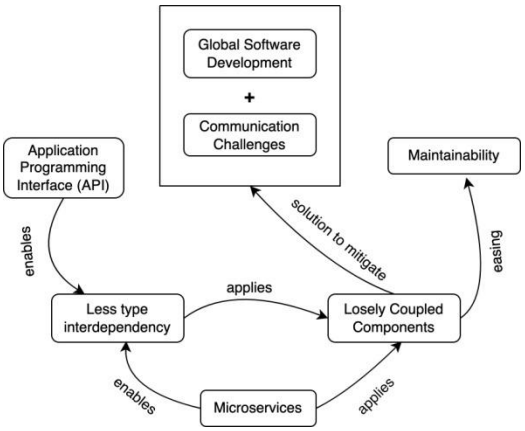

Figure 5. Losely coupled components and GSD

Sauer [SM09] recommends following some SOLID principles [43], 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 [42] architectures allow this design strategy and therefore enforce minor type interdependency.

As a possible solution to mitigate communication challenges in a GSD environment, Sievi-Korte et al. [SM10] indicates 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.

5. DISCUSSION

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. How software architecture design impacts the GSD environment?

Alzoubi and Gill [SM13] bring to our attention that the AEA [40] 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 [44]. Moreover, a mature architecture is essential to simplify transparent task distribution [2].

Sievi-Korte [SM10] highlights that API is a crucial architecting practice to define product boundaries and handle modules' limits. Component-based and APIs [45] 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 GSD teams, Mishra and Mishra [SM07] and van Vliet [46] 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 [39].

Microservices is an example of an architectural style that applies both type interdependency and loosely coupled tenets. It has become widely adopted [47], and some authors consider it reasonable to follow Conway's law [48]. 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 [49] define software component as: "*[...] a unit of composition with contractually specified interface and explicit context dependencies only. 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 [50] 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 [51] observe that adopting loosely-coupled components design is a critical factor in GSD environments to mitigate communication problems.

These findings suggest that adopting a loosely coupled software design strategy can mitigate communication challenges in GSD environments. Based on this, we built our first hypothesis: *HP1: Decoupled software architectures are communication enablers and can help to mitigate communication challenges on GSD environments.*

5.2. Is there any architectural design that can positively impact the GSD 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. [41] also support using AEA as an integrated shared view. When talking about performance on GSD teams, Alzoubi and Gill [SM13] show the contrast between adopting AEA and using EA, where no value is delivered, which the AEA definition supports [52]. However, Alzoubi et al. [SM12] indicate how AEA affects GSD teams still needs clarification. Although AEA's impact is unclear, Kornstädt and Sauer [SM05] and some authors [53] highlight adopting architecture-based development brings advantages to the GSD 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 [6], Scrum [7], and DevOps [54], may help to mitigate communication challenges which some authors also support [2, 55].

We found two perspectives when discussing architectural knowledge and knowledge management 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 [13, 56].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 hada real impact.

These findings suggest adopting agile practices combined with an architectural design focused on architecture as a decision-making process in GSD environments can help to mitigate communication challenges. Based on this, we built our first hypothesis: *HP2: An Architectural Design which enables agile practices and follows architectural-centric principles can help to coordinate GSD teams and mitigate communication challenges.*

6. 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. [57], 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 GSD 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.

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: The use of 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.

7. CONCLUSION AND FUTURE WORKS

In this work, we provide valuable insights into software architecture design in global 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 future works, we will build and execute a case study on a company with distributed teams to test the hypothesis presented in section 5 and compare the results with the finding of this study. This case study will help us to validate our findings and allow us to propose an adaptative architecture to enhance the communication aspects of the GSD team.

REFERENCES

[1] R. Camara, I. Monte, A. Alves, and M. Marinho, "Hybrid practices in global software development: A systematic literature review," *International Journal of Software Engineering & Applications (IJSEA)*, vol. 13, pp. 1–17, 2022.

[2] R. Camara, A. Alves, I. Monte, and M. Marinho, "Agile global software development: A systematic literature review," in *Proceedings of the 34th Brazilian Symposium on SoftwareEngineering*, pp. 31–40, 2020.

[3] O. Sievi-Korte, S. Beecham, and I. Richardson, "Challenges and recommended practices forsoftware architecting in global software development," *Information and software technology*, vol. 106, pp. 234–253, 2019.

[4] B. M. Yildiz, B. Tekinerdogan, and S. Cetin, "A tool framework for deriving the application architecture for global software development projects," 2012.

[5] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, pp. 1–10, 2008.

[6] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Gren- ning, J. Highsmith, A. Hunt, R. Jeffries, *et al.*, "The agile manifesto," 2001.

[7] K. Schwaber and J. Sutherland, "The scrum guide," *Scrum Alliance*, vol. 21, no. 1, pp. 1–38,2011.

[8] J. A. Highsmith and J. Highsmith, *Agile software development ecosystems*. Addison-Wesley Professional, 2002.

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

[10] S. A. Qurashi and M. R. J. Qureshi, "Scrum of scrums solution for large size teams usingscrum methodology," 2014.

[11] A. Salameh and J. M. Bass, "Heterogeneous tailoring approach using the spotify model," in *Proceedings of the Evaluation and Assessment in Software Engineering*, pp. 293–298, 2020.

[12] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIG-SOFT Software engineering notes*, vol. 17, no. 4, pp. 40–52, 1992.

[13] N. Ali, S. Beecham, and I. Mistrik, "Architectural knowledge management in global software development: a review," in *2010 5th IEEE International Conference on Global Software Engineering*, pp. 347–352, IEEE, 2010.

[14] A. Avritzer, D. Paulish, Y. Cai, and K. Sethi, "Coordination implications of software archi-tecture in a global software development project," *Journal of Systems and Software*, vol. 83,no. 10, pp. 1881–1895, 2010.

[15] V. Clerc, P. Lago, and H. Van Vliet, "Assessing a multi-site development organization for architectural compliance," in *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*, pp. 10–10, IEEE, 2007.

[16] M.-A. Vanzin, M. B. Ribeiro, R. Prikladnicki, I. Ceccato, and D. Antunes, "Global softwareprocesses

definition in a distributed environment," in *29th Annual IEEE/NASA Software En- gineering Workshop*, pp. 57–65, IEEE, 2005.

[17] R. N. Taylor, N. Medvidovic, K. M. Anderson, E. J. Whitehead, J. E. Robbins, K. A. Nies, P. Oreizy, and D. L. Dubrow, "A component-and message-based architectural style for guisoftware," *IEEE Transactions on Software Engineering*, vol. 22, no. 6, pp. 390–406, 1996.

[18] E. Newcomer and G. Lomow, *Understanding SOA with Web services*. Addison-Wesley, 2005.

[19] "What is rest ?." https://www.restapitutorial.com/lessons/whatisrest.html. Accessed: 2019-10-20.

[20] E. Wolff, *Microservices: flexible software architecture*. Addison-Wesley Professional, 2016.

[21] I. Malavolta and R. Capilla, "Current research topics and trends in the software architecture community: Icsa 2017 workshops summary," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pp. 1–4, IEEE, 2017.

[22] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *IEEE International Conference on Cloud Computing*, pp. 626–631, Springer, 2009.

[23] G. Kulkarni, "Cloud computing-software as service," *International Journal of Cloud Com- puting And Services Science*, vol. 1, no. 1, p. 11, 2012.

[24] R. Perrey and M. Lycett, "Service-oriented architecture," in *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.*, pp. 116–119, 2003.

[25] Scientific Software Development GmbH, "Atlas.ti."

[26] J. Urrego, R. Muñoz, M. Mercado, and D. Correal, "Archinotes: A global agile architecture design approach," in *International Conference on Agile Software Development*, pp. 302–311, Springer, 2014.

[27] V. Clerc, "Do architectural knowledge product measures make a difference in gsd?," in *2009 Fourth IEEE International Conference on Global Software Engineering*, pp. 382–387, IEEE, 2009.

[28] V. Clerc, P. Lago, and H. Van Vliet, "Global software development: are architectural rules the answer?," in *International Conference on Global Software Engineering (ICGSE 2007)*,pp. 225–234, IEEE, 2007.

[29] A. Kornstädt and J. Sauer, "Mastering dual-shore development–the tools and materials ap- proach adapted to agile offshoring," in *International Conference on Software Engineering Approaches for Offshore and Outsourced Development*, pp. 83–95, Springer, 2007.

[30] A. Kornstadt and J. Sauer, "Tackling offshore communication challenges with agile architecture-centric development," in *2007 Working IEEE/IFIP Conference on Software Ar- chitecture (WICSA'07)*, pp. 28–28, IEEE, 2007.

[31] V. Lenarduzzi and O. Sievi-Korte, "On the negative impact of team independence in mi- croservices software development," in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, pp. 1–4, 2018.

[32] A. Mishra and D. Mishra, "Software architecture in distributed software development: A re-view," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pp. 284–291, Springer, 2013.

[33] B. Tekinerdogan, S. Cetin, M. A. Babar, P. Lago, and J. Mäkiö, "Architecting in global software engineering," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 1, pp. 1–7,2012.

[34] J. Sauer, "Architecture-centric development in globally distributed projects," in *Agility Across Time and Space: Implementing Agile Methods in Global Software Projects*, pp. 321–329, Springer, 2010.

[35] O. Sievi-Korte, I. Richardson, and S. Beecham, "Software architecture design in global software development: An empirical study," *Journal of Systems and Software*, vol. 158, p. 110400, 2019.

[36] Y. I. Alzoubi and A. Q. Gill, "An agile enterprise architecture-driven model for geographi- cally distributed agile development," in *Transforming Healthcare Through Information Sys-tems*, pp. 63–77, Springer, 2016.

[37] Y. I. Alzoubi, A. Q. Gill, and B. Moulton, "A measurement model to analyze the effect of agile enterprise architecture on geographically distributed agile development," *Journal of Software Engineering Research and Development*, vol. 6, no. 1, pp. 1–24, 2018.

[38] Y. I. Alzoubi and A. Q. Gill, "An empirical investigation of geographically distributed agile development: The agile enterprise architecture is a communication enabler," *IEEE Access*, vol. 8, pp. 80269–80289, 2020.

[39] M. E. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[40] A. Q. Gill, *Adaptive cloud enterprise architecture*, vol. 4. World Scientific, 2015.

[41] P. Ovaska, M. Rossi, and P. Marttiin, "Architecture as a coordination tool in multi-site soft- ware development," *Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 233–247, 2003.
[42] S. Newman, *Building microservices*. " O'Reilly Media, Inc.", 2021.
[43] R. C. Martin, "Clean architecture," 2017.
[44] M. Bano, D. Zowghi, and N. Sarkissian, "Empirical study of communication structures andbarriers in geographically distributed teams," *IET software*, vol. 10, no. 5, pp. 147–153, 2016.
[45] D. Jacobson, G. Brail, and D. Woods, *APIs: A strategy guide*. O'Reilly Media, Inc., 2012.
[46] H. Van Vliet, "Software architecture knowledge management," in *19th australian conference on software engineering (aswec 2008)*, pp. 24–31, IEEE, 2008.
[47] P. Di Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: an industrial survey," in *2018 IEEE International Conference on Software Architecture (ICSA)*,pp. 29–2909, IEEE, 2018.
[48] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops," *London: Sharif University of Technology*, 2014.
[49] I. Crnkovic, "Component-based software engineeringnew challenges in software develop-ment," *Software focus*, vol. 2, no. 4, pp. 127–133, 2001.
[50] M. Page-Jones, *The Practical Guide to Structured Systems Design: 2nd Edition*. USA:Yourdon Press, 1988.
[51] J. Bosch and P. Bosch-Sijtsema, "Coordination between global agile teams: From process to architecture," in *Agility Across Time and Space*, 2010.
[52] C. Edwards, "Agile enterprise architecture, part 1," *USA: ProcessWave*, 2006.
[53] H. R. De Faria and G. Adler, "Architecture-centric global software processes," in *2006 IEEE International Conference on Global Software Engineering (ICGSE'06)*, pp. 241–242, IEEE,2006.
[54] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *Ieee Software*, vol. 33, no. 3,pp. 94–100, 2016.
[55] R. Jain and U. Suman, "Effectiveness of agile practices in global software development," *International Journal of Grid and Distributed Computing*, vol. 9, no. 10, pp. 231–248, 2016.
[56] V. Clerc, P. Lago, and H. van Vliet, "The usefulness of architectural knowledge manage- ment practices in gsd," in *2009 Fourth IEEE International Conference on Global Software Engineering*, pp. 73–82, IEEE, 2009.
[57] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, and A. Chatzigeorgiou, "Identifying, categorizing and mitigating threats to validity in software engineering secondary studies," *Information and Software Technology*, vol. 106, pp. 201–230, 2019.